Budapesti Gazdasági Egyetem
Pénzügyi és Számviteli Kar

Sebestyén Tímea

Gazdaságinformatikus

Egyedi alkalmazásfejlesztés, tesztelés és dokumentálás a Billzone.eu online számlázó alkalmazás keretében

2020

NYILATKOZAT

Alulírott ……Sebestyén Tímea…….. büntetőjogi felelősségem tudatában nyilatkozom, hogy a szakdolgozatomban foglalt tények és adatok a valóságnak megfelelnek, és az abban leírtak a saját, önálló munkám eredményei.

A szakdolgozatban felhasznált adatokat a szerzői jogvédelem figyelembevételével alkalmaztam.

Ezen szakdolgozat semmilyen része nem került felhasználásra korábban oktatási intézmény más képzésén diplomaszerzés során.

Tudomásul veszem, hogy a szakdolgozatomat az intézmény plágiumellenőrzésnek veti alá.

Budapest, 2020. év ……május…. hónap …13… nap

……Sebestyén Tímea s.k. ……..

hallgató aláírása

BUDAPESTI GAZDASÁGI EGYETEM
ALKALMAZOTT TUDOMÁNYOK EGYETEME

PÉNZÜGYI ÉS SZÁMVITELI KAR

Sebestyén Tímea

Gazdaságinformatikus

Beszámoló a szakmai gyakorlatról

2020

1149 Budapest, Buzogány utca 10-12.
Telefon: (+36-1) 469-6600
Fax: (+36-1) 469-6610
www.uni-bge.hu

Tartalom

1. A gyakorlati hely bemutatása

1.1. Cég elérhetősége

Cég név: N-Ware Informatikai és Tanácsadó Korlátolt Felelősségű Társaság
Rövid név: N-Ware Kft.
Székhely: 1139 Budapest, Gömb utca 26.
Honlap: N-Ware Kft.
E-mail: info@n-ware.hu
Telefon: 06-1-789-2379

1. ábra Az N-Ware Kft. logója
https://www.n-ware.hu/

1.2. Cég általános bemutatása

Az N-Ware Kft. egy dinamikusan növekvő, professzionális IT megoldásokat szállító és szoftverfejlesztő cég.

A vállalkozás a több éve együtt dolgozó csapatból, a Nazka Kft. hagyományait követve alakult újjá 2009 nyarán. Az N-Ware Kft-nél jelenleg 40-50 fejlesztő és tesztelő dolgozik, kapacitásukat további partnerekkel képesek növelni.

A cég fő profilja az egyedi szoftverfejlesztés és IT szaktanácsadás, különös tekintettel az online elektronikus szolgáltatások és az egyedi vállalati szoftverfejlesztés területén. Továbbá informatikai szolgáltatásuk fontos szegmense az IT outsource (informatikai munkaerő kihelyezés vagy bér-programozás) olyan cégek számára, akik nem rendelkeznek megfelelő fejlesztői kapacitással.

2. Szakmai tapasztalatok - Bevezetés

A 14 hét alatt összesen 2 projektbe tekintettem bele. A Billzone.eu online számlázó rendszer tesztelésében és dokumentálásában nyomatékosan részt vettem, és a BMW egyik leányvállalatának a weblap fejlesztésében kaptam néhány fejlesztési feladatot. Ezekből majd a későbbiekben részletesen is bemutatok egy-egy részfeladatot.

Számomra mindenképpen érdekes volt megtapasztalni a csoportmunkát és a fejlesztésben az egymásra épülő feladatok megoldását. A Billzone.eu-nál volt szerencsém kisebb javításokat is elvégezni, majd egyeztetni azzal a munkatársammal, aki utána ellenőrizte a feladatot. De az is teljesen más volt, amikor a fejlesztő munkatársam küldte nekem a megoldott feladatát, hogy teszteljem le, amikor pedig hibát találok a feladatban, akkor pontos meghatározással küldjem vissza neki a feladatot.

3. Feladataim a gyakorlat alatt

Mint már említettem, több területen is ki tudtam magam próbálni a szakmai gyakorlat alatt. Az első héten elkezdtem a betanulást, és kihívásként megkaptam a feladatot, a fejlesztői környezet kialakítását. Ez nem kis feladat volt, mert sok olyan programmal is találkoztam, amiket még nem használtam. Szerencsére a munkatársaim a segítségemre voltak. Megkaptam azt a lehetőséget is, hogy heti egyszer Homeoffice-ban dolgozzak a munkatársakhoz hasonlóan, szóval az otthoni gépemre is telepíteni kellett ugyan így a rendszert. Később nagyon szerencsésnek éreztem magam, mert így nem okozott gondot a karantén ideje alatt a munka folytatása.

A későbbiekben olyan feladatokat kaptam, amivel a rendszer működését megismerhetem. Mivel egy online számlázó rendszerről van szó, így megtanultam a rendszer használatát, felfedeztem minden weblapot benne. Ezt követően elmerültem a kódban, ahol a képeket válogattam ki. Mivel egy 10 éves rendszerről van szó, így található benne sok kép és dizájnelem, amik már nincsenek használatban például az arculatváltás miatt. Miután feltérképeztem ezt a részét is a rendszernek, kaptam még bőven apró kereső feladatot, hogy jobban megismerjem a rendszert. Amikor már bővült a tudásom róla, akkor bevezettek a dokumentálás részébe is. Elsőként a blog oldalunkat kellett átnéznem és a SEO Plugin által kijavítanom a hibákat benne. Ez által nagyon sok leírást olvastam még a számlázó működéséről és sokszínűségéről.

A harmadik héten már elkezdtem a tesztelési feladatokkal való ismerkedést. Megmutatták, mire kell figyelni, és a tesztelő oldalak használatát is. Eleinte még nem mindent értettem, szóval volt pár feladat, amit többször kellett megoldanom, de ezzel csak jobban megismertem a számlázó program széleskörű funkcióit. Nem mellesleg megtanultam helyesen és gyorsan számlázni ennek segítségével. Ezek után kaptam pár javítandó feladatot, amihez már a kód használatára volt szükségem. Megmutatták a Debugolás folyamatát és ennek segítségével kellett megkeresnem a hibákat, amiket később javítottunk is. Volt, amikor elég volt a felületen megkeresnem, de előfordult olyan is, hogy egy fél napi debugolás után sem találtam rá a hiba forrására. Ilyen esetben természetesen segítséget kaptam a munkatársaimtól, és együtt kijavítottuk.

A negyedik héten bemutatták nekem a BMW projektet is. Itt először egy SQL feladatot kellett végrehajtanom a Microsoft SQL Server Management Studio segítségével. Elsőként kissé idegen volt számomra ez a felület, de később tisztában lettem a kezelésével. Itt hasznosítani

tudtam azt a tudásomat, amit az egyetemen szereztem SQL ismeretek órán. Egy kurzor segítségével oldottam meg a problémát, némi logikus gondolkodást belecsempészve. Mindeközben a számlázó program tesztelésében is rendszeresen kaptam feladatokat.

A következő héten elérkeztünk a hónap végére, ami a számlázó program esetében az adott „Sprint" végét jelenti. Ilyenkor rengeteg tesztelést kaptunk, amit egész hónapban javítottak a fejlesztők, és elkezdtük a munkatársammal összegyűjteni mindazt a fejlesztést, amit az adott hónapban véghez vittünk. Minden hónapról a blogon közzé teszünk egy cikket, amiben összegyűjtjük a nagyobb változásokat. Ez lehet egy dizájnváltozás, vagy egy új pénznem bevezetése. Ha akkora feladat volt a hónapban, ami több magyarázatot igénylő változás, akkor arról egy külön cikket készítünk az érthetőség érdekében. Vannak olyan cikkeink is, amiket olyan esetekben tudunk használni, amikor egy felhasználónak valamilyen kérdése van a rendszer működésével kapcsolatban. Sajnos ezeket nem volt idejük módosítani minden esetben a munkatársaimnak, ezért megkaptam azt a feladatot, hogy a gyakran használt blogcikkekben a képeket gyártsam újra és cseréljem ki. Rendkívül izgalmas feladat volt ez. Sok olyan funkció működéséről tanultam ezeken keresztül, amit eddig nem teljesen értettem. A hét végén publikálásra kerültek az éles weboldalon a frissítések és javítások.

A hatodik héten főként a különböző blogcikkek létrehozásával foglalkoztam. Az anyagot már összegyűjtöttük, most viszont meg kellett fogalmazni a cikket és képeket készíteni hozzá. Itt nem csak az adott hónapban levő verziófrissítést kellett megcsinálnom, hanem még az előző évről is volt lemaradás, amit most bepótoltunk. A frissítések nem maradtak el, csak a dokumentálásra nem jutott ideje a munkatársaknak. Itt is csoportmunkában dolgoztunk hárman. A tesztelő munkatársam összegyűjtötte pontokba szedve számomra a frissítéseket, majd én elkészítettem a blogcikket és a hozzá tartozó magyarázatul szolgáló képeket, és végül miután a projektvezető átolvasta, publikáltuk azokat. A hét végére érve szembesültünk azzal, hogy az irodában csak csökkentett óraszámot tölthetünk. A vezetőség arra kért minket, hogy a munkánk 80%-át otthonról végezzük el. Később a hétvége folyamán ez még kevesebbre csökkent, mert sokaknak gyermekeik vannak és otthon maradtak az iskolák bezárása miatt. Szerencsére el tudtuk kezdeni problémamentesen az itthoni munkát.

A nyolcadik héten folytatódtak a tesztelési feladataim, amiket főként az új számlasablon tesztelésével töltöttem. Sok nyelven, különböző kombinációkban kellett a hibákat felfedeznünk. Eközben elkezdtem a BMW-nél a következő feladatomat is, amit még a karantén előtt kaptam. Ezt a későbbiekben részletesen ki szeretném fejteni, mint fejlesztői munkát.

A következő héten sok apró feladatom volt a Billzone.eu rendszerrel kapcsolatban. Elkészítettem több blogcikket is, többek között a március havi verziófrissítésről a blogcikket. A weblapok között is el tudjuk olvasni a rövidített Általános szerződési feltételeket, hogy ne kelljen a hosszú dokumentumot letölteni. Ezt a weblapot még nem töltötték fel az angol szöveggel, csak a magyar ÁSZF szöveg jelent meg ott. Nekem kellett kikeresnem a Resources fájlból, hogy melyik hivatkozásnál van ez a fordítás, majd átmásolni a rendes ÁSZF dokumentumból az angol nyelvű szöveget és megformázni HTML segítségével. A megformázás alatt azt értem, hogy a szövegeket tagolni, új sorba rendezni, félkövér és dőlt stílust beállítani, pontokba szedni a mondatokat. Ez után elkezdtem egy nagyobb volumenű feladatot, amiben az API (Application Programming Interface) dokumentálását újraírjuk. A 10 év alatt a folyamatosan bővített dokumentum már nem volt összefüggő bizonyos esetekben, ezért a projektvezető úgy döntött, hogy készítsük el a legújabb változatát, ami már nem hozzácsatolással fog működni, hanem újra generálással. Ezért az egyik munkatársam, aki ennek a számlázó programnak az alapjait készítette el, írt egy olyan programot, amivel le tudjuk generálni ezt a dokumentumot. Nekem az a feladat jutott, hogy ezt a programot feltöltsem adatokkal, amiket a régi dokumentumból veszek. Esetenként át kell fogalmaznom azt, hogy az egyszerű felhasználók számára is érthető legyen és összefüggő. Mivel az egész rendszert átfogó műveletet írunk le ebben, így eléggé terjedelmes a dokumentum.

A következő héten is ezzel kezdtem el foglalkozni, hogy az adott Sprintben publikálni tudjuk a friss információkkal feltöltött dokumentumot. Emellett még kaptam egy tesztelési feladatot, mert az egyik felhasználónk jelezte azt a hibát, hogy 0,9 €-t nem tud a számlán feltüntetni. Ezt a fejlesztő munkatársam kijavította, majd ketten leteszteltük. Másrészt volt egy feladatom, amiben a Resources fájlt kellett átírnom, mert már elavult a megnevezése a cégnek, ami fel volt tüntetve az egyik szűrőnél. Ehhez kapcsolódóan kellett még egy bővítést is bevezetnem a kódban, ahol még hozzá kellett adnom az előbb említett szűrőhöz több céget is, melyeket fel kell ismernie a rendszernek. A hét végére pedig befejeztem az API-ról szóló dokumentumot.

Elérkeztünk a hónap közepéhez, amikor az eddigi fejlesztéseinket elkezdjük mergelni a következő szintre, jelen esetben a Main ágra. Ez a hét nagyrészt a tesztelésről szól, hogy leellenőrizzük a feladatokat. Ebbe beletartozott részemről a számla létrehozása utáni befejező oldal dizájn ellenőrzése, és a saját fejlesztéseim ellenőrzése, például az Általános Szerződési Feltételek oldal fordításának az ellenőrzése. Pénteken pedig elkezdtünk már dolgozni előre a következő hónapra, melyben bevezetésre kerül majd a Díjbekérő számla típus. Ehhez

kapcsolódóan ki kellett gyűjtenem azokat a mezőket, amiket ki kell tölteni a számla létrehozása során a számlakiállítónak.

Április végéhez értünk, ezért a héten már nem sok tesztelési feladatot kaptam, viszont folytatni tudtam az API-ról szóló dokumentum következő részének az elkészítését, amiben a hibakódokat másoltam bele a kódba a magyar és az angol nyelvű dokumentum alapján. Erre azért volt szükség, mert a dokumentumot létrehozó program a kódból olvassa ki a hibakódot és a megnevezését. A rendszer összetettsége végett, több száz ilyen hibakódot tartunk számon, szóval ez több napos munka volt, mire mindet alaposan átnéztem és megformáztam. Ezen kívül kisebb hibajavításokkal foglalkoztam a héten. Ilyen például a statisztikák oldalon az egyik szűrő kijavítása, mert amikor a szűrő paramétert alaphelyzetbe szerettük volna állítani egy gomb segítségével, a dátumválasztónál a kezdő dátumot nem törölte ki.

Az utolsó két hétben a fejlesztés kapta a nagyobb szerepet a munkám során. Több olyan feladatot is kaptam, amiben a statisztikai oldalakon a szűrőknek egy-egy eleme helytelenül működött, és azt javítottam ki. Ezek legtöbbször olyan okból fakadtak, hogy nem helyes néven hivatkoztak a változóra a funkció során. Ezeket utána tesztelnem is kellett saját magamnak, de a tesztelő munkatársnak is. Ezen kívül el kellett készítenem a blogcikket az április hónapban létrehozott javításokról és új funkciókról, és a fejlesztői dokumentumokat is átnézni, hogy helyesen lettek-e a képek kicserélve a dizájnváltozás után.

3.1 Fejlesztői feladatom

Említettem, hogy 2 projektben vettem részt a szakmai gyakorlati idő alatt. Most szeretnék a BMW-s projektből is bemutatni egy feladatot, amit a ticket rendszeren alapuló óraszám könyvelését segítő alkalmazáshoz készítettem SQL segítségével.

Feladat: Készíts egy lekérdezést, ami átkonvertálja az óraszámot úgy, hogy a megadott tól-ig intervallumot egy egész számként írja ki, majd a maradék perceket tárolja el, és adja hozzá a következő naphoz, és a dolgozókra egyéni kódok segítségével szűrjön.

Ezt a feladatot a Microsoft SQL Server Management Stúdió használatával oldottam meg. Elküldték nekem, hogy a dolgozókhoz és a munkafolyamatokhoz milyen adatokat kötöttek. Ezen kívül megkaptam segítségnek azt a lekérdezést, amivel eddig dolgoztak, de egyszerűsíteni szerették volna. Hogy őszinte legyek, nem nagyon volt átlátható számomra az a megoldás, ezért arra jutottam, hogy először papíron alkossam meg az óraszámítás logikáját.

Ami könnyebb volt, hogy elsőként fel kellett töltenem az adatbázisomat adatokkal, amik természetesen nem valós adatok voltak, de a tesztelésre így is alkalmasak voltak. Ezután megalkottam a képen látható lekérdezést, amiben összekötöttem a két táblát, ahol a dolgozók óraszáma és a feladatok szerepeltek. Majd egy kurzort határoztam meg, ami azt teszi lehetővé, hogy a lekérdezés soronként ellenőrizze le a megadott feltételek szerint a táblákat.

```
DECLARE sum_hours CURSOR FOR
SELECT cast(WorkTime.dt_Start as date), SUM(DATEDIFF(minute, WorkTime.dt_Start, WorkTime.dt_Finish)) AS Minutes
    FROM [dbo].[Ticket]
    INNER JOIN [dbo].[WorkTime] ON [dbo].[Ticket].[fui_NTUserID]=[dbo].[WorkTime].[fui_NTUserID]
    WHERE WorkTime.fui_NTUserID = 1121
    GROUP BY cast(WorkTime.dt_Start as date)
 GO
```

2. ábra Saját kép; SQL lekérdezés és kurzor létrehozás

A képen látható, hogy egy lekérdezésben több összetett feladat is található, ezt szeretném kifejteni azt, részekre bontva. A CAST parancs segítségével egy bármilyen típusú értéket átkonvertálhatunk dátum típusra. A lekérdezésemben a munkaidő kezdésének a dátumát kellett venni, de a munkaidő éééé.hh.nn óó.pp.ss formában szerepel. A DATEDIFF paranccsal megkapjuk a két dátum közötti időt. Elsőként meghatározzuk, hogy percekben számoljon, majd megadjuk azt a két mezőt, amik a munka kezdés és a munka befejezés időpontját tartalmazzák. Mivel két táblával dolgoztam, ezért össze kellett kötni őket a közös pontok alapján. A WHERE feltételnél megadtuk a dolgozónak az azonosítóját, majd a GROUP BY segítségével dátum szerint csoportosítottuk, majd lefuttattuk.

A következő lépésben el kellett indítani a kurzorunkat, és megadni neki a feltételeket.

```
OPEN sum_hours;

declare @table table (id int, Datum date)

insert @table select fui_TicketID, cast(WorkTime.dt_Start as date) from WorkTime where fui_NTUserID = 1121
```

3. ábra Saját kép; Kurzor indítása

Létre kell hoznunk egy táblát, amiben meg fognak jelenni a kurzor segítségével lekérdezett adatok, majd készítünk egy lekérdezést ismét, amiben meghatározzuk a tábla tartalmát. Ezek után különböző változókat deklaráltam még, amiket a későbbiekben a számítás során használtam fel. Megadjuk a kurzornak, hogy kezdéskor a maradék perc 0, és a kurzor állása 0-ról indul, majd megadjuk a számolást a következő módon:

```
FETCH NEXT FROM sum_hours INTO @Date, @Minutes;
SET @MinutesLeft = 0;

WHILE @@FETCH_STATUS=0
BEGIN

    SET @MinutesSum = @Minutes + @MinutesLeft
    SET @Hours = ROUND(@MinutesSum / 60, 0)
    SET @MinutesLeft = @MinutesSum - (60 * ROUND(@Minutes / 60, 0))

    IF(@MinutesLeft >= 60)
    BEGIN
    SET @Hours = ROUND(@MinutesSum / 60, 0)
    SET @MinutesLeft = @MinutesSum - (60 * ROUND(@Minutes / 60, 0)) - 60
    END
    ELSE
    BEGIN
    SET @Hours = ROUND(@MinutesSum / 60, 0)
    END

    select (select id from @table where Datum = @Date) as Ticket, @Date as Date, @Hours as Hours, @MinutesLeft as Minutes

    FETCH NEXT FROM sum_hours
    INTO @Date, @Minutes;

END

CLOSE sum_hours;
DEALLOCATE sum_hours;
```

4. ábra Saját kép; Számítás megadása

Az óraszámításban a következő gondolatmenet szerint haladtam végig: első lépésben összeadom azt a percet, amit az a személy dolgozott egész nap, akire rákerestünk a SELECT parancsban a WHERE feltételnél, majd hozzáadom az egész számra kerekítés után maradt percek számát. A következő lépésben kiszámolom az óraszámot úgy, hogy az aznap dolgozott perceket elosztom 60-al, majd kerekítem 0 tizedesjegyre. Ez után meghatározom a maradék percet, amit az összes percből kivonok az egész óraszámot percekre szorozva. Ez még nem elég ahhoz, hogy a maradék percet hozzáadja a következő naphoz, és levonja mindig az elhasznált maradékot, ezért kellett egy IF metódus hozzá. Ebben elsőként megállapítjuk, hogy a maradék perc nagyobb vagy egyenlő szám-e, mint 60. Ha ez a feltétel teljesül, akkor az óraszámhoz hozzáadunk 1-et, majd a maradék percből levonunk 60-at. Ha ez nem teljesül, akkor pedig nem változik az óraszám számítása.

Utolsó lépésben ismét belefűzünk egy lekérdezést, amiben meghatározzuk az oszlopok nevét, és a beleírt tartományt. Végül lezárjuk a kurzort, majd a következő táblát kapjuk végeredményként a futtatás után.

Ebben láthatjuk, hogy a munka ticket száma melyik, amin az illető dolgozott, majd a dátumot, hogy melyik nap, és az óraszámot. A perceket csak ellenőrzés céljából írattam ki a rendszerrel, hogy le tudjam ellenőrizni a számítást.

Results Messages

	Ticket	Date	Hours	Minutes
1	2	2020-02-...	8	45

	Ticket	Date	Hours	Minutes
1	3	2020-02-...	8	5

	Ticket	Date	Hours	Minutes
1	2	2020-02-...	5	30

	Ticket	Date	Hours	Minutes
1	2	2020-02-...	7	35

	Ticket	Date	Hours	Minutes
1	2	2020-02-...	9	50

	Ticket	Date	Hours	Minutes
1	6	2020-02-...	9	0

5. ábra Saját kép; Lekérdezés végeredménye

Sajnos ez a feladat nem lett hibátlan, mert csak abban az esetben működik helyesen, amikor egy nap, egy ticketen dolgozik a munkatárs. Amikor már több tickethez van hozzárendelve, akkor elcsúsznak az órák. Ezen sajnos nem tudtam változtatni, de a munkatársaim örültek a könnyebb gondolatmenetű programnak is, amit továbbfejlesztenek majd.

3.2 Tesztelés a Billzone.eu-n

A Billzone.eu rendszernek több tesztelő felülete is van. Létezik olyan, amit csak mi, akik a projekten dolgozunk, érünk el, és van olyan is, amit a leendő felhasználók ki tudnak próbálni, hogy a cégükben működhet-e a számlázó program segítségével a számlázás.

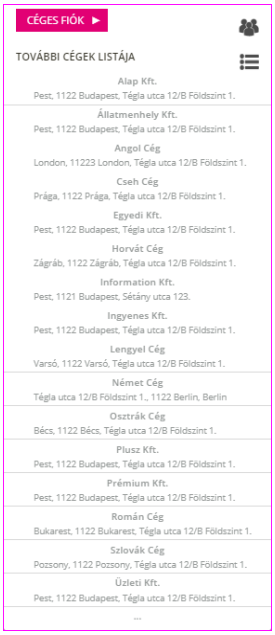
A tesztelés számunkra történhet a Visual Stúdióban való futtatással is, amivel a különböző „rétegeken” tudunk tesztelni. Ugyanez viszont létezik egy UAT (User Acceptance Test) rendszerben is, ahol szintén a 3 különböző lépcsőfokon tudjuk tesztelni annak megfelelően, hogy az adott Sprintben melyik szakaszban tartunk. Ezeket a lépcsőfokokat a DEV, a MAIN és az RELEASE ágnak nevezzük.

Minden hónapot, azaz Sprintet a DEV ágon kezdünk el fejleszteni. Ez a legalsó lépcsőfok. Ilyenkor még az éles rendszerben nem történik frissítés, csak mi látjuk a javításokat, akik ezzel dolgozunk. Amit a Visual Stúdióban dolgozunk, azt folyamatosan telepítjük az UAT rendszerekre is, hogy többen tudjuk tesztelni egyszerre a rendszert. Miután elvégeztünk minden feladatot, amit az adott hónapra beterveztek a projektvezetők, vagy éppen az aktuálisan beesett hibákat javítottuk, akkor telepítjük a DEV ágról a MAIN ágra a

frissítéseket. A telepítés után elkezdjük azokat a javításokat tesztelni, amiket eddig a DEV ágon javítottunk. Ilyenkor még kieshetnek olyan hibák, amiket eddig nem vettünk észre, és több adattal már lehet másként fog megjelenni majd a rendszerben a javítás. Az élesre telepítés előtt a RELEASE ágra is telepítve lesz a rendszer, ezzel egy időben pedig a PET tesztrendszerre is kikerül a friss verzió. Miután itt is leellenőriztük a még utána kieső hibákat, akkor a hónap végén telepítjük a frissítéseket az éles rendszerre, ahol közvetlenül a telepítés után csinálnak egy tesztet, ahol ellenőrzik, hogy hibátlanul működik-e tovább a rendszer. A felhasználók számára elérhető teszt felületet SANDBOX-nak hívjuk. Erre mindig csak akkor kerül ki a legfrissebb verzió, amikor az éles szerverre is telepítésre kerül. Ennek azért kell így lennie, hogy a felhasználók mindig azt a rendszerműködést tudják kipróbálni, ami éppen az éles rendszeren is elérhető a regisztrált felhasználóink részére. A továbbiakban 2 feladatot fogok részletesen bemutatni.

3.2.1 Tesztelés a felületen

A szakmai gyakorlat alatt legsűrűbben számlakép teszteléssel foglalkoztam. Volt, amikor az egyik ügyfelünk kért egyedi számlasablont, és azt 10 ország saját nyelvén kellett tesztelni, de volt olyan is, amikor a cég bevezetett egy új számlanyelvet, és az új nyelv miatt keletkezett formához kellett igazítani a többi nyelven a számlaképet.

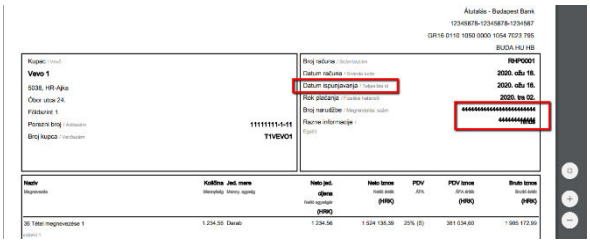


6. ábra Saját kép; A cégek listája

Ez egy roppant egyszerű feladat, hiszen csak be kell jelentkezni az egyik teszt felületünkre, ami teljesen olyan, mint amit a felhasználók is látnak az éles szerveren keresztül. Ez az UAT (User Acceptance Test) rendszer. Ezen a felületen vannak létrehozva teszt cégek, minden országból, akik számára engedélyezve van a számlázás a Billzone.eu-n keresztül. A képen is láthatjuk, hogy a tesztrendszer tartalmazza az összes országot és minden előfizetési csomagot külön-külön cégenként. A neveik alapján tudjuk beazonosítani, hogy melyik cégre van szükségünk a teszteléshez.

Egy számlakép tesztelése során úgy tudjuk legjobban kiszűrni a hibát, ha szélsőséges adatokat adunk meg. Például azt is érdemes figyelni, hogy a cég nevénél, a címadatoknál és az egyéb mezőknél a maximális karakterszámot töltsük ki, hogy meglássuk, kifér e a számlán. A kétnyelvű számlákon pedig azt kell főként figyelni, hogy a fordítás kiférjen és ne csússzon el

7. ábra Saját kép; Hibás számlakép 1.



8. ábra Saját kép; Hibás számlakép 2.

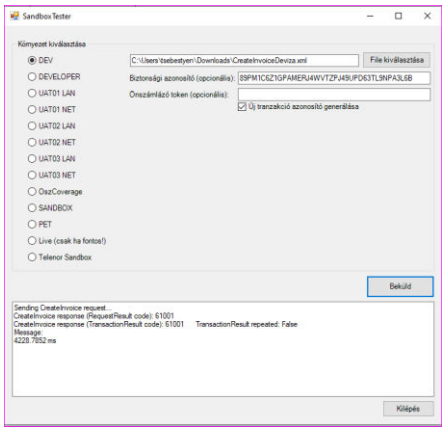
a számlakép miatta. A következő képeken a pirossal bekeretezett részeken fedeztem fel a hibákat, amiket részleteztem az imént.

3.2.2 Tesztelés a Billzone.eu tesztrendszerén keresztül

Ebben a bekezdésben szintén a számlázás teszteléséről fogok írni, de egy teljesen más módszerről, mint amit az előzőekben bemutattam. Most nem a számla kinézetét kell ellenőriznem, hanem a tartalmát.

Feladat: API-val (Application Programming Interface) számlázó cégek esetén, a számlán a bruttó összeg tartomány 0-1 és -1-0 között is szerepelhessen olyan pénznemek esetében, amiknél az összeget nem egész számként is meg lehet adni, pl. Euró.

Az egyik felhasználó jelezte felénk, hogy 0,9 €-t szeretett volna feltüntetni a számlán kedvezménynek, de hibaüzenetet kapott. Az természetesen jogos, hogy olyan esetekben, amikor csak egész szám lehet a pénzösszeg, akkor ne állítsa ki a rendszer a számlát, de olyan esetekben, amikor van váltópénze a pénznemnek, akkor engedélyeznie kellene a számla létrehozását.



9. ábra Saját kép; API tesztelő felület

Ezt kijavította a fejlesztő munkatársunk, majd a Billzone.eu saját tesztrendszerében a legkönnyebb tesztelni ezt az esetet. Ezt a tesztrendszert a Visual Stúdióból lehet megnyitni. A futtatás után egy felhasználóbarát felület jelenik meg, ahol ki kell választanom, hogy melyik tesztelési ágon szeretném lefuttatni a számla tesztelést. Be kell írnom azt a kódot, amivel a céget lehet azonosítani, majd kiválasztani az XML fájlt, amit az API magyarázó dokumentum alapján hoztak létre kizárólag tesztelési célra. Ebben a fájlban tudom megadni a különböző adatokat, amikkel tesztelnem kell, hogy a számla létrehozható legyen bizonyos esetekben is, amit az ügyfél jelzett számunkra. Az alsó

Textbox-ban a felület kiírja számunkra a tesztelés eredményét. A képen jelenleg a 61001 kód látszik, ami azt jelenti, hogy a számlát hiba nélkül létre lehet hozni. Azonban ez nem minden esetben lesz hibátlan. Ilyenkor az előbb említett dokumentumból ki tudjuk olvasni az ötjegyű kód alapján, hogy mi a hiba a számlával. Ezek a számlák a felületen is látszódni fognak annál a cégnél, akinek a nevében számláztunk.

3.3 Dokumentálás a Billzone.eu rendszerben

A Billzone.eu rendszert több módon is dokumentáljuk hónapról hónapra. Amit már többször is említettem, blogcikkek formájában is minden mozzanatunkat leírjuk a felhasználóink számára. Ebbe tartozik például a havi verziófrissítésekről a beszámoló, vagy a rendszer különböző funkcióinak a leírása lépésről lépésre. Természetesen minden információt megjelenítünk letölthető formában is, a Felhasználói dokumentumok és az Adatvédelmi tájékoztatók által. Ebben a fejezetben, szeretném ismertetni a blogcikkek készítésének a folyamatát, és az API dokumentum készítésének a folyamatát az én szemszögemből és a rám bízott feladatokból.

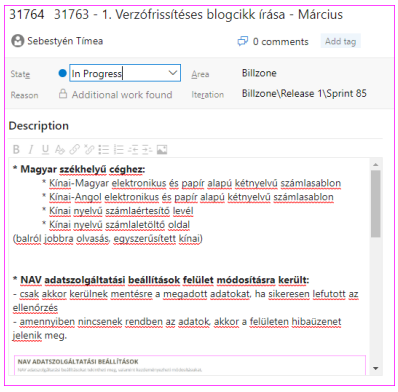
3.3.1 Blogcikk készítése, mint dokumentálás

A Billzone.eu rendszerhez kapcsolódóan több mint 300 blogcikkünk van közzé téve. A WordPress által készítjük el a különböző témájú, folyamatosan frissített cikkeinket. Ez azért nagyon hasznos számunkra, hogy a WordPress-ben dolgozunk, mert itt nagyon sok olyan funkció van, ami a segítségünkre lehet. Például, nekünk csak meg kell adni, hogy milyen kereső szavakra adja be a cikket. Ezen a felületen azt is láthatjuk, hogy egy adott időszakra lebontva, hányan kerestek rá az adott szavakra, és hányan tekintették meg a blogcikkeket, amiket publikáltunk. A szerkesztőfelülete rendkívül felhasználóbarát, és sokban hasonlít a Word szerkesztőre.

10. ábra WordPress oldal logója https://fontmeme.com/wordpress-font/

Feladat: Írjunk meg egy blogcikket a március havi frissítésekről!

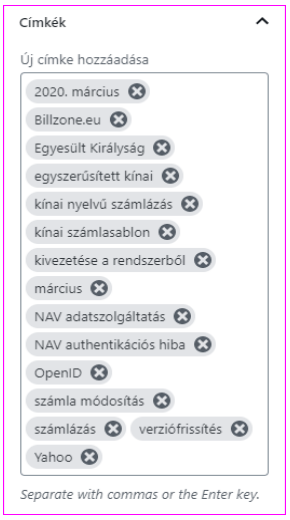
Elsőként megnézem, hogy a feladataimnál mik azok a részek, amiknek szerepelnie kell a blogcikkben. Erre van a cégnek egy saját rendszere, ahol nyilvántartjuk a feladatainkat, és a projekten belül minden munkatárs látja egymás feladatát, és hogy éppen hol tart benne. Nekem már kigyűjtötte a havi frissítéseket a munkatársam, hogy könnyebb dolgom legyen:



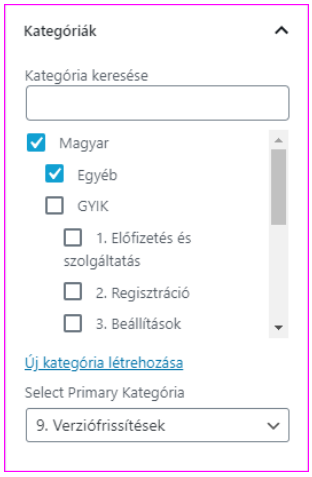
11. ábra Saját kép; A feladat leírása

Ez után, megnyitom a WordPress oldalt, ahol bejelentkezek a Billzone.eu oldal fiókjába. Értelemszerűen kiválasztom azt, hogy „Új hozzáadása”, majd kiadja a cikk szerkesztőfelületét, ahol külön be tudom állítani a címet, és a fő szöveget. Természetesen mindent meg kell tenni annak érdekében, hogy a blogcikkünk érdekes legyen, szóval érdemes különböző képekkel színesíteni azt, amin láthatják, hogy ténylegesen hol találjak a beállításokat a rendszeren belül. Ezeket a képeket általában az UAT rendszerben készítem el, ahol a legfrissebb verzió található. Ha olyan cikket készítek, ami az akkori működést szeretné leírni, akkor mindig azon az UAT rendszeren tesztelem, ami megegyezik azzal a verzióval, ami az éles szerverre van telepítve.

Miután megírtam minden fontos információt a cikkben és képekkel szemléltettem azt, akkor hozzárendelek még különböző keresőszavakat, amik a témához kapcsolódnak, és besorolom egy téma alá, amikkel általában foglalkozni szokott a blogunk. Ezeket a szerkesztéseimet egyelőre még nem publikálom, mert a projektvezetőm és a tesztelő személy átolvassa, hogy biztosan mindent helyesen írtam le. Esetenként még hozzátesznek gondolatokat, vagy kivesznek belőle, ha túl soknak ítélik a magyarázatot, és utána ők teszik közzé a kész blogcikket a weblapon.



13. ábra Saját kép; Címkék megadása



12. ábra Saját kép; Kategóriák megadása

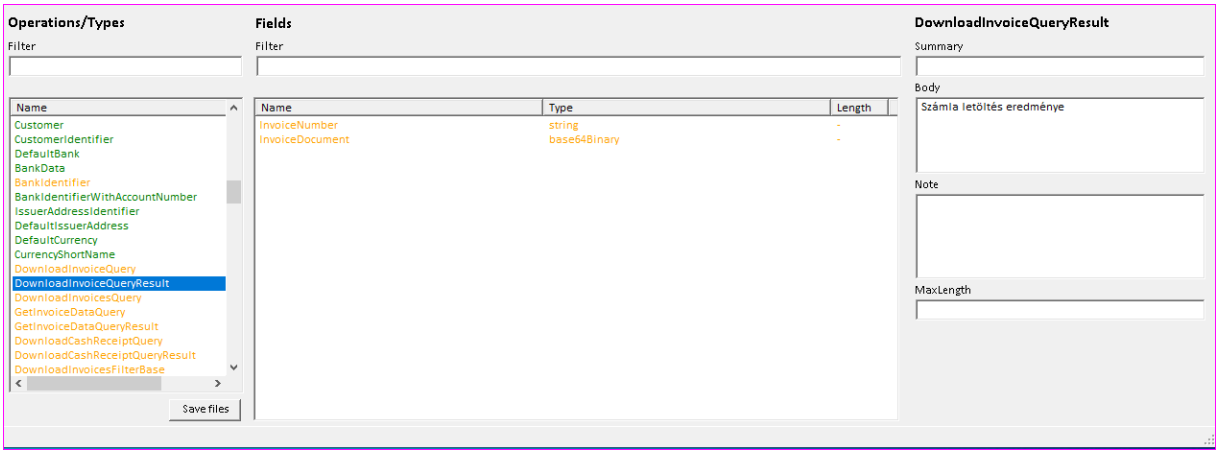
3.3.2 API rendszer leírása dokumentumban

A Billzone.eu rendszerben sok helyen használjuk az API (Application Programming Interface) rendszert. Többek között arra szolgál, hogy a NAV adatszolgáltatást összekösse a Billzone számlázó rendszerrel, így ennek segítségével nem kell külön bevinni a számla adatait, mert automatikusan átküldi a NAV részére azokat a számlákat, amiknek az ÁFA tartalma meghaladja a 100.000 Ft összeget. Ezt az API rendszert használják még arra is, hogy a webshopokat összekössék a számlázó rendszerekkel, így létrejöhet az azonnali számlakiállítás, amihez nem szükséges az eladónak beavatkoznia, csak a vevő kiválasztja az adott termékeket, majd a rendszer önmagától generálja a számlát.

Természetesen ez csak 2 példa arra, hogy mit hozhatunk létre az API segítségével. Ezért hoztunk létre egy dokumentumot, aminek a segítségével magyarázatot kapnak a felhasználók a működésére. Mivel már ez a dokumentum 10 éves volt, és mindig csak bővítve lett, úgy döntött a vezetőség, hogy csináljunk egy teljesen friss és érthető másolatot róla. Ehhez csinált az egyik fejlesztő egy programot, hogy a fő címekhez csak hozzá kelljen csatolni a kiegészítő szövegeket.

Feladat: Illeszd be az adott címek mellé a szöveget úgy, hogy az a felhasználók számára teljesen érthetően legyen megfogalmazva.

Elsőként áttanulmányoztam az eredeti dokumentumot, hogy számomra is érthető legyen minden, és átlássam a kapcsolatokat a címek között. Utána megnyitottam egy .cmd fájlt, ami elém tárt egy teljesen felhasználóbarát programot, amiben oldalt láttam a címeket, középen pedig textbox-ok voltak, ahová nekem kellett beillesztenem a megadottak szerint a szövegeket. Közben nekem a kulcsszó szerint az eredeti dokumentumban ki kellett keresni, hogy melyik címhez melyik bekezdés kapcsolódik, majd ezt beilleszteni a programba. Mivel ez több napos munka volt, így részletekben kellett elmentenem a Visual Studioba, hogy közben a munkatársam le tudja ellenőrizni az eddig felvitt adatokat.



14. ábra Saját kép; Dokumentum generáló program

A képen látható a dokumentum generáló program, aminek a segítségével beillesztem egymáshoz csatolva a fő szövegét a dokumentumnak. Bal oldalt látjuk a címeket. Ami zöld színnel van írva, azokhoz már kitöltöttem minden szükséges adatot, hogy létrejöjjön a dokumentumban a bekezdés. Ami még sárgával van, az természetesen még nincs szerkesztve. Középen látható a két sorból álló táblázatunk, amit ugyanezen struktúra szerint kell létrehoznunk. Jobb oldalra kell a szövegeket beilleszteni. Tudunk egy címet adni neki, amiről szól az adott bekezdésünk. A Body részbe írjuk azt a szöveget, ami megmagyarázza a cím

jelentését a felhasználók számára. Ehhez kapcsolódóan tudunk megjegyzést is írni a Note tagba. Amikor pedig a táblázatról van szó, akkor meg tudjuk adni, hogy mekkora lehet a maximális hossza a karakterláncoknak, amiket a weblapon be lehet illeszteni az adott helyre. A képen például azt a két sort tartalmazza a táblázatunk, ami a számlák sorszámát generálja, és a számla letöltést követően a letöltött dokumentum nevét hozza létre. Ez a két mező is összeköttetésben van egymással. Ezeknél nem szükséges MaxLength-et megadni, mert a generátor tudja magától, hogy milyen hosszúnak kell lennie.

Mindezek után, rányomunk a felületen található „Save Files” gombra, ami elmenti a változtatásunkat, és megjeleníti a Visual Studio alkalmazásban a 2 fájlt, amin változás történt. Ezek után egy sorszám alapján regisztrálnom kell a munkámat a céges rendszerbe, hogy a munkatársaim számára is elérhetővé váljon a változtatásom. Ezt a műveletet minden Visual Studios változtatásunk után el kell végeznünk, így működhet az, hogy egy feladatot több ember is tud ellenőrizni, vagy hogy a projekten több ember is tud egyszerre különböző feladatokat megoldva dolgozni.

4. Összefoglalás

Összességében úgy látom, hogy a gyakorlati helyen megfelelően el tudtam sajátítani minden fontos lépését egy alkalmazás fejlesztésének, tesztelésének és dokumentálásának. A cégnél minden alkalommal amikor kérdésem volt, bárkihez fordulhattam, mindenki szívesen segített nekem. Beletekinthettem a csoportmunkába, és a feladatok összetettségét is megfigyelhettem. Megtapasztaltam, az önálló munkavégzés fontosságát, és hogy hogyan kell csoportmunka esetén jól és hatékonyan kommunikálni a munkatársaimmal. Úgy érzem, hogy a munkafolyamatban sikeresen a munkatársaim segítségére voltam, és sok esetben a kezük alá tudtam dolgozni, hogy nekik időt spóroljak. Örömömre szolgált, hogy amikor már betanultam, akkor önálló hibajavításokat is rám bíztak a weblap fejlesztés során. Ezekből is azt éreztem, hogy a Billzone.eu fejlesztő csapat teljeskörű munkatársa vagyok.

5. Képek

BUDAPESTI GAZDASÁGI EGYETEM
ALKALMAZOTT TUDOMÁNYOK EGYEMEME

PÉNZÜGYI ÉS SZÁMVITELI KAR

Sebestyén Tímea

Gazdaságinformatikus

Egyedi alkalmazásfejlesztés, tesztelés és dokumentálás a Billzone.eu online számlázó alkalmazás keretében

2020

1149 Budapest, Buzogány utca 10-12.
Telefon: (+36-1) 469-6600
Fax: (+36-1) 469-6610
www.uni-bge.hu

Tartalom

1. Bevezetés

Záródolgozatom fő témája az alkalmazásfejlesztést fogja körülölelni. Bemutatásra kerül ezen belül a fejlesztés folyamata, dokumentálása és tesztelése mind általánosan és mind a munkahelyen tapasztaltak által.

Az alkalmazásfejlesztés egy forráskódból épül fel, amit egy úgynevezett fordítóprogrammal tudunk futtatni, értelmezni. A futtatás során a program elsőként egy köztes nyelvre fordítja le magát, majd a buildelt köztes nyelvről fordítódik a gép által értelmezhető nyelvre. A C# nyelv a .NET keretrendszer által támogatott, egyik legnépszerűbb programozási nyelv, ami közeli kapcsolatban áll a Java-val.

Tanulmányaink során elsőként a Java nyelvvel ismerkedtünk meg, ami azt a célt szolgálta, hogy könnyebben tudjunk programozni a későbbiekben objektumorientáltan a C# nyelv elsajátításával. A nyelv eredete már a nevéből is könnyen kikövetkeztethető. A C és a C++ nyelvek után fejlesztették ki, mert egy könnyebben kezelhető programozási nyelvet kerestek, ami az MS Visual Basic-hoz hasonlít, de a C és a C++ nyelv precízségével rendelkezik. „A nyelv mottójának talán a következő képletet tekinthetjük: A Basic egyszerűsége + a C++ hatékonysága = C#!” (ILLÉS Z. 2005. P. 11). Végül a Microsoft által újonnan létrehozott fejlesztői környezetben, a Visual Studio.NET programcsomagban került bemutatásra a C# 2002-ben. Ezt a nyelvet kimondottan a .NET keretrendszerre alkották meg. A nyelv fontosabb elemei a következők:

- Neumann-elvű, akár rendszerprogramok írására is alkalmas.
- Több fordítási egységből áll, de minden egységnek megegyezik a szerkezete.
- Egy sorba több utasítás is írható, de az utasításokat pontosvesszővel le kell zárni.
- A függvénydefiníció esetén nem használható a blokkstruktúra.
- A függvények és változók neveiben használhatunk ékezetet, és a kis- és nagybetűk között különbséget teszt a nyelv.
- Egyszeres öröklést alkalmaz, interfészek használatával.
- Könnyen definiálhatók a párhuzamosan végrehajtott szálak.

Végeredményképpen létrehoztak egy könnyen kezelhető és dinamikus programnyelvet, amit nem kell több órán keresztül írni, ha egy futtatható programot szeretnénk kapni. Személy szerint otthonosnak érzem a Visual Studiot is, mindent könnyen meg lehet benne találni, némi angol tudás segítségével.

2. A gyakorlati hely és a téma kapcsolata

A gyakorlati helyemet az N-Ware Kft-nél töltöm. A cég fő profilja az egyedi szoftverfejlesztés és IT szaktanácsadás, ezért több nagy céggel állnak hosszú idő óta partneri kapcsolatban. Például a BMW, a Siemens, és a T-Systems Magyarország Zrt is szerződésben áll velük. Én a szakmai gyakorlatom ideje alatt 2 projektben vettem aktívan részt. Az egyiknél inkább a szoftverfejlesztést tapasztalhattam meg, a másiknál pedig a tesztelést és a dokumentálást gyakoroltam. Azért választottam témának az alkalmazásfejlesztést, mert a Billzone.eu rendszeren keresztül minden lépést részletesen be tudok mutatni a tapasztalataim alapján.

2.1 Billzone.eu online számlázó rendszer

A Billzone.eu egy online számlázó rendszer, amit a T-System Magyarország Zrt forgalmaz, és az N-Ware Kft fejleszt. Ezt a szolgáltatást 2010-ben dobták piacra, akkor még az N-Ware Kft volt a tulajdonosa, majd 2018-ban vette meg a T-Systems Magyarország Kft. A fejlesztést viszont továbbra is a cégünk viszi tovább, mert egy komoly fejlesztő csapat áll mögötte. Az évek során akkora lett az igény az online számlázásra, hogy már nem csak Magyarországon lehet a Billzone.eu-val számlázni, hanem számos európai országban is jelen van, naprakész jogi háttérrel. Ne tévesszen meg bennünket az, hogy online számlázó rendszer. Ezzel nem csak elektronikus számlát lehet kiállítani, hanem papír alapút is, de a környezetvédelmi okokat nézve többen használják az elektronikus változatot. Erről a rendszerről azt kell tudni még, hogy sok számlázó rendszerrel ellentétben, ezt nem szükséges letölteni minden olyan eszközre, amivel a cég számlát szeretne kiállítani. Internetelérés segítségével minden eszközön alkalmazható, elég hozzá egy felhasználói fiókot létrehozni.

15. ábra A Billzone.eu online számlázó logója https://kosarertek.hu/ceg/billzone-eu/

2.1.1 Az elektronikus számla

A 2010/45/EU direktíva fogalma szerint e-számlának minősül az a számla, amely a kiállítás és a befogadás pillanatában elektronikus. Azonban mind az említett direktíva, mind a magyar jogszabály (ÁFA törvény) kimondja, hogy ugyanakkor az e-számlának teljesítenie kell az alábbi alapelveket: megbízható módon biztosítani kell a számla eredetének hitelességét, adattartalma sértetlenségét és olvashatóságát.

Mind az említett EU direktíva, mind a magyar ÁFA törvény a már meglévő és az új hitelesítési lehetőségek mellett továbbra is az egyik megbízható hitelesítési módnak a digitális aláírást javasolja.

A hatályos jogszabályokban elvárt megbízható ellenőrzési nyomvonal létesítése érdekében a Billzone.eu elektronikus számlái eredetiségét egy hitelesítő szolgáltató által kiállított tanúsítvánnyal való elektronikus aláírás biztosítja. Ezen felül a Billzone.eu e-számla egy időbélyegző szolgáltató által kiállított időpecséttel kerül kiállításra, így garantálva a kiállítás dátumának valósságát.

3. Alkalmazásfejlesztés

Manapság, ha azt a szót mondjuk, hogy alkalmazás, akkor mindenki az okostelefonokra letölthető szoftverekre asszociál. Sokan nem gondolják, hogy amit a számítógépre is le lehet tölteni, vagy az interneten lévő összes weboldal, sőt, még a böngésző is egy alkalmazás. Ezeket természetesen fejleszteni kell, mivel az informatika napról napra fejlődik. Mindig kitalálnak egy új dizájnelemet, vagy egy új kódot, de abban is fejlődnek, hogy a betöltése az alkalmazásoknak gyorsabb legyen.

Az alkalmazásfejlesztés lépéseit főként a C# nyelvben szeretném bemutatni. A program szerkezete rendkívül összetett, és tetszőleges számú modulból tevődhet össze. Ezeknek a moduloknak az összességét projektnek szoktuk nevezni, a modulokban összefoglalt osztályok összességét pedig programnak hívjuk. A modulokat egy névtér segítségével tudjuk összekapcsolni, aminek a segítségével tudunk hivatkozni egy-egy metódusra, ami ismétlődik a programban. A nyelv a kis- és nagybetűt külön karaktereknek definiálja, ezért nagyobb a mozgásterünk az elnevezések szempontjából. Lehet két ugyanolyan nevű függvényünk, de ha az egyik kisbetűvel a másik pedig nagybetűvel kezdődik, akkor már két külön metódust határoztunk meg vele. Természetesen vannak olyan kulcsszavak és védett azonosítók, amikkel nem definiálhatunk másokat, mert azoknak más szerepe van a nyelvben, így például a break, bool, string, public szavak és még sok másik nem lehetnek elnevezések, mert nem tudja azokat értelmezni a program fordítás közben. Fontos, hogy megjegyzéseket is írjunk a programunkba, főként, amikor több fejlesztő is dolgozik egy adott projektben. Ezt két módon is megtehetjük: ha több soron keresztül szeretnénk megfogalmazni a megjegyzést, akkor a /* és */ jelek között kell elhelyeznünk a magyarázatot. Ha nem sok a mondandónk vele kapcsolatban és elfér egy sorban, akkor elég a sor elejére a // jeleket írni, és már írhatjuk is a

megjegyzésünket. Ebben az esetben nem kell lezárnunk a megjegyzés rovatot, mert a nyelv felismeri, hogy csak arra a sorra vonatkozik.

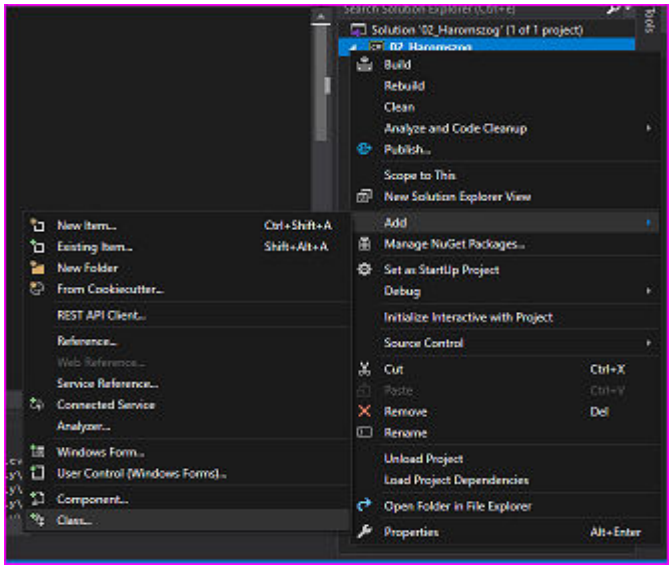
Amikor eldöntöttük, hogy írunk egy programot, és meghatároztuk, hogy milyen nyelven szeretnénk megírni azt, akkor több módunk is van az elkezdéshez. Ha nem áll rendelkezésünkre olyan program, amibe írni is tudjuk a programot és fordítani is tudjuk egyaránt vele, akkor elkezdhetjük egy sima jegyzettömbbe is, vagy egy dokumentum fájlba. A mentésnél viszont oda kell figyelnünk, hogy a formátum az adott nyelvhez kapcsolódó formátum legyen. Ezután a parancssorba meg kell hívnunk egy adott paranccsal, és már láthatjuk is a végeredményt. A C# nyelvben ezt úgy kell megtennünk, hogy a szövegszerkesztőben megírt fájlt .cs formátumban mentjük el, majd a parancssorban a következő módon hívjuk meg: csc példa.cs, majd rányomunk az enter gombra. A fordítás kimenetele a példa.exe állomány lesz.

A leggyakrabban használt IDE (integrált fejlesztői környezet) a Visual Studio. Ebben meg tudjuk írni a kódot is, és fordítani is tudjuk azt, mindössze egy gomb megnyomásával. Amikor elindítjuk a programot, ki kell választanunk, hogy új projektet szeretnénk létrehozni, vagy egy meglévőben szeretnénk tovább dolgozni. Ha az új projekt létrehozását választjuk, akkor meg kell adnunk a lehetőségek közül, hogy milyen nyelven és milyen formában szeretnénk megírni a programunkat, és el kell neveznünk a projektet is. Ha egy projekt több modulból áll össze, akkor a Solution Explorer-ben tudunk elnavigálni közöttük.

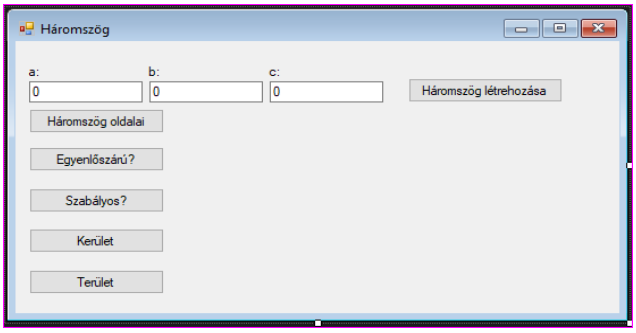
16. ábra A Visual Studio logója
https://visualstudio.microsoft.com/

Ez a fejlesztőfelület grafikus alkalmazások fejlesztésére is alkalmas. Itt meg tudjuk tervezni is a weblapot egyszerűbb elemekkel, és a hozzájuk tartozó funkciókat is egyszerűen meg tudjuk adni. Az egyetemi Objektum Orientál Programozás nevű tárgyon belül megtanultuk, hogyan hozzunk létre egyszerűbb alkalmazásokat mind kinézetre, mind működésre. Ezt szeretném bemutatni egy feladaton keresztül. A feladat pedig a következő: Készítek egy programot, amiben megadjuk a Form-on keresztül a háromszög oldalinak hosszát, majd a program egy gomb megnyomására kiírja, hogy melyik oldalnak milyen értéket adtam meg, majd további gombok megnyomása után megvizsgálja a program, hogy szabályos-e és egyenlőszárú-e a háromszög és kiszámolja a kerületét és a területét.

Elsőként hozzunk létre egy Háromszög nevű osztályt, amiben a program értelmezi a Textboxba írt számok értékét. Ezen kívül hozzuk létre a metódusokat, amik meghatározzák a háromszög szabályosságát és hogy egyenlőszárú-e, majd a kerület és terület képletet. Ezeknek a metódusoknak találó nevet adjunk, amiket majd a program további részében meghívunk. Az osztályt pedig a következő módon lehet hozzáadni: jobb klikkel rákattintunk a projekt nevére, majd a kurzort az Add feliratra irányítjuk, és az utána felugró résznél a Class…-ra kattintunk. Elnevezzük az osztályt, és utána kezdhetjük a program írását.



17. ábra Saját kép; Osztály létrehozása



18. ábra Saját kép; Háromszög form megjelenése

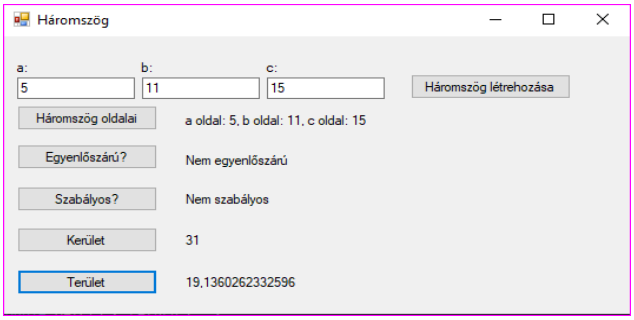
Utána készítsük el a Form felületet, ahol meg tudjuk adni a háromszög oldalainak hosszát és gombnyomásra megjelenítse az adatokat, amiket kérünk tőle. Ehhez szükségünk lesz a Toolboxra, amiben ki tudjuk választani a megfelelő elemeket, például a gombot, a szövegdobozt és a címkét. Ezeknek egyenként adjunk nevet, amire hivatkozunk a Háromszög osztálynál, hogy a program futtatásakor el tudják látni a funkciójukat. A képen láthatjuk, hogy a programban már deklaráltuk az a, b, c változókat, és kezdő értéknek nullát adtunk meg.

```
public Haromszog(double aa, double bb, double cc)
{ a = aa;
    b = bb;
    c = cc;
}
1 reference
public bool Egyenloszaru()
{ if (a == b || a == c || b == c) return true;
    else return false;
}
1 reference
public bool Szabalyos()
{ if (a == b && a == c) return true;
    else return false;
}
1 reference
public double Kerulet()
{ return a + b + c; }
1 reference
public double Terulet()
{ double s = (a + b + c) / 2;
    return Math.Sqrt(s * (s - a) * (s - b) * (s - c));
```
19. ábra Saját kép; Metódusok az osztályban

A Háromszög osztályon belül leírt metódusokat a mellékelt képen láthatjuk. Ezért kellett adnunk minden metódusnak egy találó nevet, hogy a Form.cs osztályban a gombnyomáshoz tudjuk őket kapcsolni. Ezt megoldhatjuk úgy is, ha a gombra a Form-on duplán klikkelünk, majd a megjelenő metódusban adjuk meg a számolási feltételeket, de ha egy nagyobb rendszert készítünk, akkor célszerűbb az a megoldás, hogy a

főosztályba írunk meg minden logikával kapcsolatos függvényt, amit a későbbiekben a neve alapján meghívunk.

A feladat futtatásakor elsőként meg kell adnunk a háromszög oldalainak a hosszát, és megnyomni a „Háromszög létrehozása” gombot. Következő lépésben kiírja, hogy melyik oldalnak milyen értéket adtunk meg, ha megnyomjuk a „Háromszög oldalai” gombot. Minden gombnyomás után kiírja a rendszer számunka az általunk megadott számokból kiszámított terület, kerület értékeket, és megvizsgálja a feltételek szerint, hogy egyenlőszárú és szabályos háromszöget adtunk-e meg.



20. ábra Saját kép; Program futtatásának az eredmény

3.1 A Billzone.eu alkalmazás fejlesztése

A Billzone.eu online számlázó rendszer alapjait Szlaukó Zoltán alkotta meg. Ez egy rendkívül összetett rendszer, ami lehetővé teszi a felhasználók számára a hivatalos számlakiállítást, és a Nemzeti Adó- és Vámhivatal részére a hatályos jogszabályi előírások szerint online adatszolgáltatást végez a rendszerben kiállított számlákról. Ezen kívül találhatóak benne statisztikai oldalak is, amik a felhasználót abban segítik, hogy visszakeressék a kiállított számláikat, vagy hogy nyomon tudják követni, hogy egy hónapban/évben mennyi számlát állítottak ki számlatömbökre bontva.

Az alkalmazás több nyelvből áll össze. Fő nyelve a C#, ez alapján van megírva minden olyan mozzanat, amivel az alkalmazásnak dolgoznia kell, például a bejelentkezés a rendszerbe, a számla kiállítás folyamata, a számla beküldése a NAV részére. A programhoz tartozik a Resources nevű .xlsx fájlt is, amiben minden belső oldalnak a nyelvesítését, a fontos funkciókat hívó (pl számlakiállítás) oldalak szövegét is tároljuk. Ezen kívül van egy nagy adatbázis, ami szintén hozzá van csatolva a fő kódhoz. Ebben tároljuk például a felhasználók adatait, a számlaképek feliratait, és a kiállított számlákat sorszám szerint, és az előbb említett resources.xlsx fájlból telepítéskor beemeljük az adatokat ebbe az adatbázisba, majd az adatbázisbeli táblából hívja meg a rendszer a szöveget.

Az előzők alapján látható, hogy ez az online számlázó rendszer egy nagyon összetett projekt, szóval egy egész csoport dolgozik rajta. Ebben szerepel 2 fejlesztő, akik a működésért felelősek, 2 dizájner**,** velem együtt 2 tesztelő, akik a 4 fejlesztőnek a munkáját teszteljük napi

szinten és minden tesztfelületen. Ezen kívül van egy termékfelelős munkatársunk, és egy supportos munkatársunk. Ők természetesen az ügyfelek igényeivel és a rendszerrel kapcsolatos kérdéseikkel foglalkoznak, továbbá átadják az ügyfelek kéréseit és a hibákat a fejlesztőknek.

Ahhoz, hogy létre tudjon jönni a csoportos munka, az egyik legnépszerűbb verziókezelő és projekt menedzselő programot használjuk, aminek az Azure DevOps Server a neve. Ennek a rendszernek van egy saját adatbázisa a céghez rendelve egyedileg, amiben számontartják Ticketek segítségével a feladatokat. A rendszert régebbi nevén TFS-nek hívjuk, ami a Team Foundation Server rövidítése. Ez azt teszi lehetővé, hogy egy fájlban egyszerre többen is tudjunk dolgozni, viszont, ha egy bizonyos részletet egyszerre módosítottak, akkor adódhatnak gondok a mentés során, és akkor újra kell mergelni a rendszert. Ennek segítségével nyomon tudjuk követni a fejlesztéseket, és hogy milyen változtatást, ki és mikor végzett a kódon a feladathoz kapcsolva.

Természetesen, mielőtt munkába tud állni valaki, előtte létre kell hozni a fejlesztői környezetet, ami sok program telepítéséből áll. Elsőként a Visual Studio legfrissebb változatát kell a gépünkre installálni. Ehhez kapcsolódóan több funkcióját is hozzá kell rendelnünk még, hogy a rendszer összetett nyelvezetét tudja fordítani. Ezen kívül szükségünk lesz még a Microsoft Build Tools-ra, az MSSQL-re, SQL Management Studiora, egy friss Office csomagra és még néhány beállításra a számítógépen.

A Billzone.eu fő programozási nyelvei a C# és a Javascript. A Javascript sokban hasonlít a C# nyelvre, mivel nagyjából a C nyelv szintaktikáját alkalmazza. Ezek olyan részletekben rejtőznek, mint például, hogy az utasítások külön sorba kerülnek és egyértelműen el vannak választva egymástól. Az itt megjelenő adattípusok viszont már nem feltétlenül egyeznek meg a C# nyelv típusaival. Ebben például nem különbözteti meg a nyelv az egész és a tört számokat, hanem egyben number-ként kezeli őket, de a szöveges adatokat szintén string formában dolgozza fel. Ez egy gyengén típusos nyelv, ami annyit jelent, hogy minden változónak van típusa, de amikor a változót deklaráljuk, akkor nem minden esetben szükséges megadnunk az adat típusát, viszont amikor megadjuk a változó értékét, azzal megadjuk a típusát is. Megjegyzéseket ugyan azon módon tudunk létrehozni, mint a C# nyelvben, ami annyit takar, hogy ha egy sorban szeretnénk elhelyezni, akkor // jellel kezdjük az adott sort, vagy a kódba folytatólagosan írjuk a // jelet és mögé a megjegyzést, és ha több soron keresztül tart ez a megjegyzés, akkor az első sorban a /* jelet majd az utolsó sorban a */ jelet alkalmazzuk a mondat végén. Ebben a nyelvben szintén alkalmazunk kódblokkokat is, ami

egyszerűen megfogalmazva a kapcsoszárójelek közé írt részt jelenti, például az IF, FOR, WHILE utasításoknál.

Az előbb említett két nyelv főként az oldal back-end részét fedi le, és minden logikáját ebben írjuk meg. Viszont kellett létrehozni neki egy front-end részt is, amit a felhasználók vizuálisan láthatnak a weblap felkeresése során. Ezeket az elemeket HTML, CSS és Javascript nyelvek segítségével hozták létre. Főként ASP.NET Web Forms Framework-ön alapszik a Front-Endünk, amiket speciális .cshtml formátumú fájlokban tárolunk el, ezzel létrehozva a C# és HTML nyelv kombinációját.

A HTML nyelv megjelenik minden weboldal készítése során. A weboldal végső formáját a böngésző segítségével alakítjuk ki, mivel az értelmezi a megjelenítést szabályozó elemeket. Az elnevezését a HyperText Markup Language rövidítéséből kapta, ami magyarosítva a hiperszöveg leírónyelv nevet viseli. A nyelv szabványosítását a World Wide Web Consortium (W3C) független szervezet végzi. Jelenlegi legfrissebb verziója a HTML 5.0, amely megfelel az SGML (Standard Generalized Markup Language) dokumentumszabványnak is. Ezek az elemek a tag (ejtsd: ''teg'') elnevezést kapták, amiket ellátunk különböző tulajdonságokkal, attribútumokkal. Ezeket mindig szögleteszárójelek közé írjuk. Ezeknek általában mindig van egy kezdő és egy záró tag-je, és a kettő között elhelyezett adatot fogja megformázni a tag-ben megadott tulajdonságokkal.

A CSS nyelv a HTML nyelv kibővítésére szolgál. Ennek segítségével az összes oldalunknak professzionálisabb megjelenítést adhatunk, mindössze egyetlen CSS fájl létrehozásával. Ennek segítségével egy stílusbeli változást nem kell minden lapon megismételnünk, hanem elég egy helyen formáznunk, és automatikusan az összes lapon frissülni fog. Az oldalon lévő elemeket stílusmegadással tudjuk formázni, amit később a böngésző jelenít meg. Fontos, hogy az oldalainkat több böngészőben is teszteljük, mert nem minden elem kompatibilis minden böngészővel. A CSS nyelvben meg tudunk adni minden formázási lehetőséget, amit normál esetben például egy szövegszerkesztőben is meg tudunk adni. Ebbe beletartozik a szövegek formázása, képek formázása, de a háttér és a méretezések is. De az elemek elhelyezését a HTML szerkesztésénél kell meghatároznunk, ebben kell hivatkoznunk a stílusosztályokra, amiket létrehoztunk.

3.2 Miért egyedi ez a fejlesztés?

Amiért egyedinek tudnám mondani a Billzone.eu fejlesztését, az a rengeteg technológia megjelenése egy rendszeren belül. Ki szeretném emelni ezek közül részletesebben az MSSQL

Server fontosságát és szerepét, a JQuery technológiát, amit a Web Forms-ban alkalmazunk, és az ASP.NET alkalmazását a rendszerben, a WCF (Windows Communication Foundation) szerepét a számítógépek összeköttetésében. Ezen kívül szeretnék magyarázatot adni egy fontos táblázatunknak, amit Resources-nak nevezünk, a rendszerrel való összekötésére.

Egy SQL Serverre azért van szüksége a rendszernek, mert minden adatot, ami a programhoz tartozik, tárolnunk kell. Ebbe beletartozik a számlasablonok szövege különböző nyelvekre fordítva, a rendszerbe regisztrált cégek adatai, a felhasználók által létrehozott számlák adatai, a biztonságosnak mentett IP címek listája, és még sok más. Ezeket többféleképpen is be tudjuk vonni a kódunkba. Meghívhatjuk azt, az SQL-ben adott név alapján, vagy pedig megírhatjuk a kódban a lekérdezést, a megfelelő táblákra hivatkozva. Az utóbbihoz kapcsolódóan volt több feladatom is a munka során, amikor a rendszerben meghívott SQL lekérdezést kellett bővítenem, hogy a statisztikai szűrők az admin oldalon megfelelően működjenek. Ehhez létrehoztak egy osztályt, amiben összekötik az SQL szervert a kóddal. Ezzel a megoldással működik a már említett Resources fájl összekötése is. Munka közben egy Excel fájlban dolgozunk, amibe be vannak illesztve a legördülő listáink adatai, vagy a számlasablon és a weboldalak nyelvesítése is. Ez azonban át van vezetve egy adatbázisba, ahol szintén hasonló táblaként van elmentve, mint ahogy mi az Excel fájlban nyomon követjük. Erre létrehoztak szintén egy beolvasó funkciót, és annak a segítségével vezették be a rendszerbe. Ez az úgynevezett Key-k alapján köti a kódhoz a szövegeket. Ezeket a Key-ket mi az Excel táblában is látjuk, és ennek segítségével tudjuk javítani az adatokat.

A JavaScript egyik legnépszerűbb keretrendszere, a JQuery, ami a Billzone.eu fejlesztésében nagy szerepet kap. Előnyeként tudom feltüntetni, hogy könnyebbé teszi a HTML dokumentumok bejárását, és számtalan bővítmény elérhető hozzá kapcsolódóan. Ennek segítségével az oldalunkon animálást, és többféle stílust tudunk megjeleníteni. Megkönnyíti a Front-Endes munkatársak dolgát is abban, hogy ennek a használatával könnyebben lehet a mobiltelefonokhoz igazítani az oldalunkat. De például az a funkciója is hasznos, hogy a dokumentumokból egy CSS szelektor segítségével könnyedén ki tudja válogatni az adatokat, amiket a művelet után belehelyez egy tömbbe.

A .NET keretrendszer azért kerül be a programunkba, mert ennek segítségével egy olyan futási környezetet tudunk kialakítani, amivel a komplex és robosztus programok írása könnyebb és gyorsabb lesz. Az egyik legnagyobb előnye, hogy platformfüggetlen, amit egy felügyelt kód segítségével hoztak létre a CLR-ben (Common Language Runtime, ami a programozási nyelvek közös futtatórendszere). Ezt a műveletet az operációs rendszer felett

futtatjuk le. Ezt követően minden CLR-re alkalmas eszköz a forráskódot CIL (Common Intermediate Language, ami egy általános köztes nyelv) formátumra alakítja át, de ez a kód egy eszközön sem futtatható le, szóval az úgynevezett JIT (Just In Time) fordításra is szükségünk lesz. Ez a fordító alakítja át a CIL kódot az eszközön futtatható kódra. Ez által válik a .NET keretrendszer minden eszközön futtathatóvá. A .NET keretrendszer működéséhez minden esetben szükségünk lesz úgynevezett assembly létrehozására. Ezeket magyarul szerelvényeknek nevezzük. Ebben tárolja a programunknak az adatait és az erőforrásait, ez által csak olyan kódot hajt végre a keretrendszer, amik assembly-ben vannak tárolva. Az assembly egy gyűjtemény, amiben a program forráskódját egy vagy több EXE vagy DLL fájlban tároljuk el. Ez tartalmaz egy úgynevezett manifeszt-et is, amiben leírják adatok formájában a kódot és az erőforrásokat. Ezek lehetnek privát illetve publikus assembly-k is, amikben a privát assemblyt csak a saját alkalmazásunk használhatja, a publikus assembly esetén pedig a kódot mindenféle alkalmazás használhatja. Az első esetben természetesen csak mi módosíthatunk a kódban, a másodiknál pedig vigyáznunk kell, mert bárki bármikor felülírhatja a programunkat. Amikor rengeteg assembly települ a gépünkre, és mind például egy TextBox-ot használ, akkor minden assemblyhez tartozik egy DLL fájl. Azért, hogy ezt a sok fölösleges adatot kikerüljük, a publikussá tett assembly-ket a keretrendszer egy globális assembly-gyorsítótárba tölti fel. Ezeknél fontos odafigyelni az elnevezésre, mert a legtöbb hibaforrás az elnevezésből adódik. Ezért szükséges az assembly-ket egyedi azonosítóval ellátni, erre használják az erős és a megosztott neveket. A rendszerben a régebbi verziójú assembly-k kezelése is megoldott, nincs akadálya a több verzió telepítésének. A memóriakezelés sok nyelvben problémát okoz. A .NET keretrendszer viszont a használat utáni elfelejtés elvét támogatja, amit a Microsoft alkalmazott a CLR létrehozásakor. A keretrendszer fejlesztését 1998-ban kezdte el a Microsoft, amiben a programozási nyelveket össze szerette volna vonni egy közös virtuális géppel és nyelvi könyvtárral.

A WCF (Windows Communication Foundation) szolgáltatás-orientált architektúrájú programokat alkalmaz a .NET keretrendszeren belül, amivel az API (Application Programming Interface) felületeket köti össze egymással. Ezeket végpontokon keresztül tudja összekapcsolni egymással, viszont nem csak egy végponton keresztül tudnak a WCF-hez kapcsolódni. Ennek segítségével tudjuk a Billzone.eu online számlázó rendszerünket különböző webshopokkal összekötni, amikben létrejön az önszámlázás. Önszámlázásnak azt a folyamatot nevezzük, amikor nem a szolgáltatást teljesítő, azaz eladó, állítja ki a számlát a

vásárlás során, hanem a vevő a termékek összeválogatásával létrehozza a számlát a termékek kosárba helyezése után. Természetesen a számlán fel kell tüntetni, hogy ez a számla „önszámlázással” jött létre. Ennek ellenére a számla kiállítójaként a szolgáltatást nyújtó fél adatait kell szerepeltetni a számlán. A Billzone.eu oldalon ez a funkció nem minden felhasználó számára elérhető. Az egyik feltétele, hogy olyan előfizetési csomagban legyen a cég, ami tartalmazza az API funkció használatát. Ezen felül egy külön megállapodást is kell kötni, amiben feltüntetésre kerül az önszámlázás funkció használata az adott cég esetében.

4. Dokumentálás

Egy program kialakításánál nagyon fontos szerepet játszik a dokumentálás. Abban az esetben is, ha csak egy személy dolgozik a projekten, de akkor is, ha egy egész csoport dolgozik a program használhatóságán.

Dokumentálásnak hívhatjuk a programon belül létrehozott úgynevezett komment vagy megjegyzés részeket is, amiben leírjuk, hogy az adott metódus mire szolgál a programban. Ezzel abban is segíthetünk, hogy ha egy bizonyos ideig nem foglalkoztunk azzal a részével a programnak, akkor a megjegyzésben látjuk, hogy milyen funkciót szolgál ez az egész programot tekintve, és így nem fogjuk kitörölni véletlenül sem a metódust a rendszerből. Továbbá, ha új munkatársat vonunk be a projektbe, akkor ezek alapján fel tudja térképezni a rendszer működését, és láthatja, hogy melyik függvény mire utal, vagy melyik részletre visz tovább a program futása közben. Ezt Debugolással is könnyen el tudjuk végezni. Természetesen nem árt, ha külön leírást is készítünk a programunkról, mert értékesítésnél szükségünk lesz egy olyan dokumentumra, amiben beszámolunk a program működéséről. Ha ezt a program írása közben tesszük meg, akkor megkönnyíthetjük a saját munkánkat vele, mert utólag nehezebb kifejteni a gondolatainkat a kód alapján.

A programokban nem ritka, hogy a megjegyzéseket úgy készítik el, hogy azt már egyből egy XML fájlba menti ki a fordító program. „ A C# fordítónak ha a /doc:fájlnév formában adunk egy fordítási paramétert, akkor /// karakterek utáni információkat XML fájlba (a megadott névvel) kigyűjti.”(ILLÉS Z. 2005 P. 18) Ezzel kapcsolatban nekem is volt a gyakorlati idő alatt feladatom, amikor az API (Application Programming Interface) leírásáról készült dokumentációt frissítettük, de erről bővebben a szakmai gyakorlati részben írok.

Az egyik legfontosabb dokumentum, a felhasználói dokumentum. Ezt általában a fejlesztő, a betanító és a felhasználó szokta alkalmazni. Ebben szerepelnie kell a program feladatának, amivel megkönnyítheti az életünket. A futási környezet leírásában fel kell tüntetnünk a

számítógép információit, mint például az operációs rendszert, memóriaméretet, grafikus kártyát és a perifériaigényt. Természetesen le kell írnunk a program részletes használatát, és minden hozzákapcsolódó kérdést, ami a felhasználókban felmerülhet. A hibaüzeneteket és a hozzájuk tartozó leírást és a hibák lehetséges okát is mindenképpen le kell írnunk.

A felhasználói dokumentáció sokban hasonlít a fejlesztői dokumentálásra. A fejlesztői dokumentáció viszont azoknak szól, akik a hibákat keresik a rendszerben tesztelés során, vagy hibákat javítanak és fejlesztik a rendszert. Ebben le kell írnunk minden specifikációt és a követelményeket, a futási és fejlesztési környezetet, amibe beletartoznak a programozási nyelvek is. Feltüntethetünk még benne különböző teszteseteket is, amiben leírunk néhány jellegzetesen felvitt adatot és a várt eredményt. Tehetünk javaslatokat is, amikben megfogalmazzuk a fejlesztők számára, hogyan tudják megkönnyíteni a saját munkájukat, ezzel gyorsabbá és hatékonyabbá téve a munkájukat.

Különböző programismertető dokumentálásokat is alkalmazhatunk, amikben megfogalmazásra kerül a vásárló számára, hogy miért vásárolja meg az általunk fejlesztett programot. Leírja a program feladatát és tulajdonságait. Esetenként néhány hardver és szoftver elemet is megjeleníthetünk benne, melyekre szüksége van a programnak a hibamentes futáshoz. Ha egy nagyobb terjedelmű programról van szó, akkor nem árt, ha van installálási kézikönyv is, ami megkönnyíti a felhasználó számára a telepítési folyamatokat, és ezáltal nincs szükség külön szakemberre, akinek installálni kell a programot minden felhasználó számára, aki igényt tart rá.

És hogy ezeket milyen formátumban illik megadni? Programonként változó, de a legtöbb esetben PDF (Portable Document Format) formátumot alkalmaznak. Ennek az az egyik legnagyobb előnye, hogy a felhasználók számára csak olvasható formátumban adjuk át a dokumentumot. Ezen kívül, ami a nevében is megjelenik, hordozható formátum. Jelen esetben azt jelenti a hordozhatóság, hogy különböző felületekről megnyitva nem tapasztalunk elcsúszást a képek és a szövegek terén, és megnyitható minden eszközzel, ami minimum egy fajta böngészőt tartalmaz. Természetesen ez olyan programokra nem feltétlenül érvényes, amiket például egy CD segítségével juttatunk el a felhasználó számára. Ezekhez vannak olyan esetek, amikor még papír alapú használati útmutatót és installálási útmutatót csatolnak, de legtöbbször már a CD-n jelenik meg a használati és installálási útmutató PDF formátumban a program telepítője mellett. Ha viszont egy weboldalról van szó, akkor ezeket a dokumentumokat a letölthető forma mellett megtekinthetjük egy külön oldalra másolva, mintha egy PDF-et megnyitnánk, csak letöltés nélkül. Ennek az az előnye, hogy könnyebben

bele tudják csempészni az oldal design részét, amivel színesebbé tehetik az olvasók számára a dokumentumokat.

4.1 A Billzone.eu dokumentálása

Több módon is dokumentálásra kerül a Billzone.eu program fejlesztése, mivel szeretnénk értesíteni a felhasználóinkat is a fejlesztésekről. Ezért frissítjük minden hónapban a 2 részre bontott Felhasználói dokumentációnkat, aminek az egyik része a Számlakiállítás és kapcsolódó funkciók címet kapta, a másik rész pedig a rendszerhasználatról szól. Ezeket azért szedtük külön részekre, mert nagyon összetett a működése az oldalnak, és ha valaki ebből szeretne tájékozódni, akkor könnyebben megtalálja az információkat amire szüksége van. Természetesen létezik a weboldalnak egy Általános Szerződési Feltételek dokumentációja is, hogy a felhasználói jogokról tájékoztassuk ügyfeleinket magyar és angol nyelven egyaránt. Ebbe bele van foglalva az úgynevezett Süti funkciók működése is. A rendszer használ API (Application Programming Interface) segítségével is funkciókat, tehát erről is létrehoztak egy összefoglaló dokumentumot, aminek a legfrissebb verziójának a létrehozásában én is tevékenyen részt vettem. Ezen kívül még rengeteg dokumentum található meg az oldalon, ami különböző cégeknek szól, akikkel szerződésünk van, és kisebb lényeges funkciók leírása is megtalálható itt.

Aki szeretne egy kicsit könnyebb megfogalmazásban olvasni a rendszer működéséről és fejlesztéseiről, az felkeresheti a Billzone.eu rendszer saját blog oldalát, ahol naprakész információkkal tudunk szolgálni az ügyfeleknek. Itt minden hónapban részletezzük, képekkel színesítve a fejlesztéseket, az új funkciókat és a dizájnváltozásokat is. Ide készítünk egyes kisebb funkciókról lépésről-lépésre leírást, hogy könnyebben tudjanak számlázni vagy éppen statisztikai adatokat letölteni az ügyfelek. Tájékoztatást adunk a Support ünnepi üzemeléséről is, hogy a felhasználók ne aggódjanak, ha nem kapnak rögtön választ a leveleikre vagy a hívásukra. Mindez nem jöhetne létre, ha nem lenne a WordPress weboldal, aminek segítségével percek alatt összeállíthatunk egy blogcikket, és egyből publikálni is módunkban áll ennek segítségével.

A WordPress egy felhasználóbarát blogszerkesztő felület, aminek a segítségével különböző képszerkesztő funkciók is elérhetők számunkra. Ebben a rendszerben nem csak manuális vezérléssel hozhatunk létre dizájnelemeket, hanem át tudjuk váltani HTML nézetbe is a

felületet, amiben például remekül lehet táblázatot létrehozni és minden adatot vele kapcsolatban megadni. Ezen keresztül tudjuk még szabályozni, hogy milyen keresőszavakra jelenjen meg a blogcikkünk a különböző böngészőkben, és ez által azt is nyomon követhetjük, hogy hányan keresték fel az adott cikket. Böngészőkre bontva azt is meg tudjuk nézni, hogy melyik keresőszavakra, hányadik helyen adta be a Billzone.eu oldalt a kereső. Ennek segítségével megállapíthatjuk, hogy eléggé frappáns keresőszavakat használunk-e, vagy változtassunk a koncepciónkon. Azért ajánlom ezt az alkalmazást egy weboldal, vagy jelen esetünkben a blog készítéséhez, mert a beállításai pontosan ügyelnek minden részletre. Figyelnek arra, hogy reszponzív legyen a weblap, azaz minden eszközön szétcsúszás nélkül olvasható legyen a felület. Azt is előtérbe helyezik, hogy minden böngészőben megjelenjenek az általa használt formátumok. Egyszerű, és könnyen kezelhető felületről van szó, ami testre szabható.

A szakmai gyakorlatom során nagyon sok alkalommal volt szerencsém használni ezt a rendszert. Elsőként a SEO Plugin beállításokkal foglalkoztam. A SEO Plugin (Search Engine Optimization) a keresőoptimalizálásért felel. Pontosan figyel a blogcikk címére, annak is a hosszára és tartalmára, beállíthatunk kategóriákat és keresőszavakat is. Nem mellesleg, egy blogcikk létrehozásakor nekünk nem kell kézzel létrehoznunk egy külön oldalt, amit akkor dob fel az oldal, amikor a blog oldalon a címre kattintunk, hanem önmagának generál egy oldalt. Ide kapcsolja majd a témában hozzá tartozó cikkeinket is, és engedélyezi a témákról való keresést a blogon megadott címek alapján.

A Billzone.eu weblapnál azért is fontos szerepet tölt be ez a blog, mert sok olyan cikket hoztunk létre, amivel jelentősen megkönnyíthetjük a supportért felelős munkatárs munkáját. Vannak gyakran ismételt kérdések, amikre létrehoztunk számos blogcikket, és a munkatársnak elég válaszban elküldeni azt az URL-t, ami a cikkhez tartozik. Természetesen kiegészíti a saját gondolataival és a felhasználó egyedi kérdésére is válaszol. De ha egy funkciót nem megfelelően tud használni a felhasználó, magyarázatul szolgálnak a cikkek és képekkel szemléltetik a lépéseket.

4.2 Miért egyedi ez a dokumentálás?

Azért tekintem egyedinek ezt a dokumentálást, mert például a blogcikkek által a felhasználóink végig tudják nézni a számlázó fejlődését a képek által. Ebből láthatják, hogy a blog 2012-es indulása óta, milyen fejlődés volt a rendszerben. Megjelenik minden dizájnbeli változás, az átmenet az N-Ware-es és a Telekomos üzemeltetés között, amikor még zöld és

rózsaszín dizájnelemek keveredtek, és a számlakiállítás fejlődése is megfigyelhető. Ezt nem minden weboldal mondhatja el magáról, hogy ennyire nyomon követhetőek a funkciók és a dizájn fejlesztések egy blog által. Ezek az apró részletek nem figyelhetők meg a felhasználói dokumentumokban, ugyanis ott mindent naprakészen frissítünk, a funkciókat, a javításokat és a hozzájuk tartozó képeket. Erre a feladatra most kezdtünk el egy új módszert alkalmazni, hogy a képeket ne manuálisan kelljen frissíteni minden dizájnváltozáskor, hanem ha felülírjuk a képeket, akkor önmagát frissítse a dokumentum egy frissítő makró segítségével.

A blog frissítésével nincs ilyen könnyű dolgunk. Már említettem az előző fejezetben, hogy vannak olyan cikkek, amik a működésről szólnak, és azokat időnként manuálisan tudjuk csak frissíteni. Ezt azért könnyebbnek is tartom olyan szempontból, hogy nem a közel 100 oldalas dokumentumot kell igazgatni, mert a kép kivágása után elcsúszik, majd az új kép beillesztése után ismét formázást igényel, a blogon pedig csak beszúrjuk a régi kép helyére az új képet, és nem keletkezik a blogcikkben hiba. Megvan számára a bekezdés, ahová csak be kell illeszteni a friss képet, és megadni a méreteket, majd a kép alá írt szöveget. Azért tudom összehasonlítani ezt a két dokumentálási folyamatot, mert mind a két dokumentummal dolgoztam a szakmai gyakorlat során többször is.

5. Programtesztelés

Egy Program tesztelésére főként azért van szükségünk, hogy felfedezzük benne a hibákat. Hibák minden programban keletkeznek a kód írása során, mivel emberek készítik, és az emberek tudnak hibázni. Viszont a teszteléssel felfedezett hibák javításával növeljük a termék minőségét, és megbízhatóságát. De az sem kizárt, hogy a tesztelés után nem marad benne hiba. Ezért kell azokat a funkciókat a legfigyelmesebben és akár többször is tesztelnünk, amiket a felhasználók a legtöbbször fognak alkalmazni a programunkban.

A tesztelésnek 7 alapelve van:

1. A tesztelés jelzi a hibák jelenlétét. Képes jelezni a hibákat, de azt nem, hogy ha már nincs egyetlen hiba sem.
2. Kimerítő tesztelést nem lehetséges véghez vinni, ugyanis nem tudunk minden bemeneti kombinációt tesztelni, és nem is éri meg. Rendszerint csak a legkockázatosabb és magas prioritású részeket teszteljük.
3. Korai tesztnek nevezzük, amikor a rendszert már a legminimálisabb formájában is teszteljük. Ez azért jó, mert minél korábban találunk meg egy hibát, annál olcsóbb lesz a javítása.

4. Hibák csoportosulásának nevezzük azt a jelenséget, amikor az időnk végessége miatt csak azokat a részeket teszteljük le, ahol a legvalószínűbb a hiba kiesése.
5. Féregirtó paradoxon: Ez akkor következik be, amikor a fejlesztés során mindig ugyan azokat a teszteseteket futtatjuk le, akkor egy idő után már nem jeleznek hibát, hanem alkalmazkodnak a rendszer működéséhez, de a hiba ott marad.
6. A körülményektől is függ a tesztelés. Másképpen tesztelünk egy olyan programot, amit egy nagy cégnek fejlesztünk, és máshogy, amit egy iskolai dolgozathoz készítünk. Nem mellesleg nem mindegy, hogy pár óránk van a tesztelésre, vagy pedig több napunk.
7. A hibátlan rendszer csak egy téveszme. Ha a megrendelőnek nem tudjuk kielégíteni az elvárásait a rendszerrel kapcsolatban, akkor ha minden hibát ki is javítottunk, akkor sem mondhatjuk hibátlannak, ha a felhasználónk számára nem használható. (http://aries.ektf.hu/~gkusper/SzoftverTeszteles.pdf DR. KOVÁCS L. 1.1 FEJEZET)

A tesztelés során különböző technikákat tudunk alkalmazni. A két fő technika, az úgynevezett feketedobozos (black-box) és a fehérdobozos (white-box) technika. A feketedobozos tesztelés más néven specifikáció alapú tesztelés, amikor a tesztelő nem látja a forráskódot, és a specifikáció alapján tesztel. Ehhez a teszteléshez szükségünk van a lefuttatott szoftverre, leggyakrabban egy adott bemenetre tudjuk, hogy milyen kimenetnek kellene születnie, és összehasonlítjuk a várt eredményt a kapott eredményünkkel. A fehérdobozos technológia során a tesztelő látja a forráskódot is, ezt strukturális tesztelésnek nevezzük. Általában a forráskód alapján készítik el a tesztesetet. A struktúra, azaz lefedettség, pedig megmutatja a tesztelők számára, hogy a rendszer hány százalékát fedi le a tesztelés. Ezzel legtöbbször kódsorokat, elágazásokat, osztályokat, metódusokat és funkciókat tesztelünk.

A tesztelés szintjei:

- Komponensteszt: A rendszer egy komponensét teszteli csak. Ezt hívjuk másnéven fehérdobozos tesztelésnek is. Gyakori fajtái: unit-teszt és a modulteszt.
- Integrációs teszt: Ez a teszt már 2 vagy több komponens együttműködését, és az összekötő interfészeket teszteli.
- Rendszerteszt: Már a nevében is benne van, hogy a rendszer egészét teszteli le, megvizsgálja, hogy megfelel-e a követelmény és funkció specifikációknak és a rendszertervnek.
- Átvételi teszt: Ezt a tesztet általában a végfelhasználók végzik. A legismertebb fajtái közé tartozik az alfa, béta, felhasználói átvételi teszt és az üzemeltetői átvételi teszt.

A tesztelési tevékenység viszont rendkívül összetett folyamat. Nem csak a tesztek lefuttatása tartozik bele a munkakörbe, hanem például a teszttervek elkészítése, tesztesetek tervezése, végrehajtása, az eredmények értékelése és mindezekről jelentést is kell készíteni. A teszteset elkészítését is több részre tudjuk bontani. Célja, hogy egy meghatározott vezérlési út során tesztelje a programot, vagy egy meghatározott követelmény teljesülését ellenőrzi. Végrehajtása során a rendszerben a megadott kezdő állapotban kell tesztelni, a megadott input értékek alapján, majd a teszt futási eredményét össze kell hasonlítanunk a várt eredménnyel. A teszt forgatókönyvének is hívhatjuk a specifikációt, ami leírja a teszteset végrehajtásához szükséges lépéseket. Ezeknek a teszteseteknek és specifikációknak az összességét pedig tesztkészletnek hívjuk, amit csoportosíthatunk egy vizsgálati hibára vagy egy teszt alanyra. Ebből adódóan, megfelelően kell archiválnunk ezeket a dokumentumokat, ugyanis a fejlesztés során többször előkerülhet egy teszt.

A teszt tervezési technikáknak több fajtáját is alkalmazhatjuk. A specifikáció alapú teszteket a rendszer specifikációja alapján hozzák létre, ezt más néven Black-box technikának is nevezik. A modell alapú technika lényegében az előző csoport egyik tagja, de ez közvetlenül az UML modell alapján hajtja végre a teszteket. A struktúra alapú technika, azaz White-box technika a kód ismerete alapján létrehozott tesztelést foglalja magában. A gyakorlat alapú technika a tesztelők saját tapasztalatait veszi alapként.

5.1 Tesztelés a Billzone.eu rendszerben

Mivel maga a rendszer is rendkívül egyedi az összetett funkcióiból adódóan, így a tesztelése sem lehet hétköznapi. Természetesen tartalmaz minden elemet, amit egy rendszer tesztelése során alkalmaznunk kell, de mégis megvan minden részletnek a maga különlegessége. Nem mindegy, hogy egy számlát hogyan tesztelünk a különböző előfizetői csomagokban, és nem mindegy, hogy melyik számlasablont teszteljük. A számlakiállítás funkcióját is néznünk kell, mert van sima számla, van proforma azaz díjbekérő számla, előleg számla, sztornó, módosító vagy jóváíró számla. Ha azt nézzük, ezt összességében számlázási funkció tesztelésnek hívjuk, de mégis sok különböző tulajdonság alapján kell a szempontokat figyelembe vennünk a tesztelés során.

A rendszernek számos tesztelési lehetősége van. Rendelkezésünkre áll több saját tesztelési oldal, illetve a kód változtatása után a Visual Studioban is érdemes tesztelni. A leendő felhasználóink számára is létrehoztunk egy tesztrendszert, ahová be tudnak regisztrálni, mint az éles rendszerbe, és minden funkciót ki tudnak próbálni, hogy számukra megfelelő-e a

számlázó programunk. Ez azért jöhetett létre, mert egy online számlázóról beszélünk, amihez nem szükséges a számítógépre letölteni különböző programokat, amik segítségével számlát tudnak kiállítani a felhasználóink. Mindenki számára elérhető, csak internet hozzáférésre van szükség hozzá.

Elsősorban a program fejlesztése során a Visual Studioban teszteljük le a programrészletet, amit kijavítottunk. Ez azért jó, mert már ekkor kieshetnek hibák, és nem küldözgetjük oda-vissza a tesztelő munkatársunkkal a feladatot, hanem addig tudjuk korrigálni, amíg mi jónak nem látjuk. Utána természetesen elküldjük tesztelésre, és akkor a tesztelő munkatárs is leteszteli a helyi rendszeren, majd feltelepíti a legfrissebb verziót az UAT (User Acceptance Test) szerverre, ahol másik felhasználókkal, esetleg másik országokból regisztrált cégekkel is ellenőrizni tudja a javított részeket. Ezek mellett tudni kell, hogy egy adott Sprintben, ami a cégnél egy hónapot jelent körülbelül, több ágon is teszteljük a javításainkat. Elsőként a DEV ágon dolgozunk, és az ehhez tartozó UAT01 rendszeren. Két-két és fél hét után feltelepítjük a frissítéseket a MAIN ágra, ahol ismét tesztelések következnek. Ilyenkor még kieshetnek olyan hibák, amiket az éles rendszerre való telepítést megelőzően javíthatunk, hogy ne a következő telepítésnél essenek ki ezek. Ehhez is tartozik egy UAT02 nevű tesztfelületünk. A hónap utolsó pénteki napján hajnalban pedig kikerülnek a javítások az éles rendszerünkre, ehhez az UAT03 tesztoldal tartozik. Telepítés után elsőként a smoketeszt lesz végrehajtva, amivel azt nézhetjük meg, hogy minden fontosabb funkció rendben újra el tudott-e indulni a pár órás leállás után. Ilyenkor frissítjük a SANDBOX rendszerünket is, amin a leendő felhasználóink tudják tesztelni a rendszert. Ez azért csak havonta egyszer van frissítve, mert tartania kell ugyan azokat a funkciókat, amivel az éles rendszerünk is éppen rendelkezik. Ebből a SANDBOX rendszerből sem csak egy van a Billzone.eu oldalához kapcsolva, ugyanis vannak olyan cégek, akik szoros megállapodásban állnak a cégünkkel, és egyedi igényeik voltak a számlázóval kapcsolatban. Nekik külön létrehoztunk egy tesztfelületet, ahol meg tudják tekinteni az általuk jelzett hibákat és kéréseket a javítás és fejlesztés után. Ilyen szerződésben állunk például a WebEye Telematics Group-al, vagy a First Businesspost Kft.-vel és természetesen a Billzone.eu forgalmazó partnerével, a Magyar Telekom Nyrt.-vel.

Vannak olyan funkciók is a rendszerben, mint például az API (Application Programming Interface) rendszer, amivel szintén számlákat tudunk kiállítani, például webshopok számára. Ennek a számlakiállítását nem tudjuk a sima rendszeren keresztül letesztelni, de a számlázó megalkotója létrehozott egy tesztelő felületet, amibe be tudjuk illeszteni a céghez tartozó azonosítót, és az XML alapú számlát, majd kiadja a végeredményt egy kód formájában. Ezt a

kódot az API rendszerhasználatot ismertető dokumentumban tudjuk megtekinteni, hogy lássuk a hibakód jelentését. Ezen dokumentum alapján hozták létre a különböző számlákat is, hogy a megfelelő formátumból tudjon a rendszer tesztelni. Ebben meg tudjuk határozni a számlasablon nyelvét, a vásárló adatait, és a számlán szereplő tételeket. A céghez tartozó kód alapján a számlakiállító adatait automatikusan beilleszti a rendszer magának.

Természetesen nem csak számlakiállítást kell tesztelnünk attól függetlenül, hogy ez egy számlázó rendszer. Vannak különböző statisztikai oldalaink is, amikkel meg tudják tekinteni például egy időszakra vonatkozóan, hogy hány számlát állítottak ki, és még egyéb szűréseket is végre tudnak hajtani vele. Ezeket a szűrőket is rendszeresen kell ellenőriznünk, mert a hónapról hónapra való frissítés során elcsúszás történhet a rendszerben. Ezen kívül ellenőrizni kell azokat a törzsadat felvevő oldalainkat is, ahol meg tudjuk adni a számlatömböket, a vevőinket, akiknek rendszeresen állítunk ki számlát, és a termékeinket, amikhez meg tudjuk adni az egységárat és megjegyzést is. Bármilyen változtatást egyszerűen lehet véghezvinni a rendszerben. Legtöbbször olyan munkatársak is szokták tesztelni a javításokat, akik nincsenek benne minden nap a számlázó fejlesztésének a munkálataiban, így ők is észlelhetnek hibákat, amiken a tesztelők átugranak, mert régóta így működik a rendszer, csak nem fedezte még fel senki hibaként. Ezért volt az is segítség a fejlesztők számára, amikor engem bevontak a munkába, mert én teljesen új szemmel néztem át ezeket a hibákat a rendszer működésében. Természetesen nekem is meg kellett tanulnom minden lépést, hogy hogyan működik a rendszerben a számlakiállítás, de amikor már belejöttem, nehezebb teszteseteket is nyugodtan rám mertek bízni.

6. Összegzés

Összességében úgy gondolom, hogy a lehető legalkalmasabb programozási folyamatban tudtam részt venni a gyakorlati helyem által, ugyanis volt benne informatikai rész is, de szintén kamatoztatni tudtam a gazdasági területen szerzett tudásomat. A szakdolgozatomban részletesen be tudtam mutatni minden munkafolyamatot, ami egy alkalmazás fejlesztése során felmerülhet.

Elsőként a céget és a téma kapcsolatát részleteztem. Az N-Ware Kft egy szoftverfejlesztő cég, akik 10 éve dolgoznak együtt egy folyamatosan fejlődő csapattal. Manapság már 50-60 főt foglalkoztat a cég, több alcégével együtt. Az alkalmazásfejlesztést, mint témát azért is választottam, mert tudtam, hogy a munkahelyen ezt a témát fogom a gyakorlatban alkalmazni, tehát erről tudok a legrészletesebben írni. A cégen belül sikerült egy olyan projektet választani

a felsőbb vezetőimnek, ami tökéletesen beleillik a gazdaságinformatikusok szakmájába. Tartalmazott a munkám programozási és informatikai feladatokat is, de hasznosak voltak a számlázás során a pénzügyi ismereteim is. Amikor a dokumentálás részben vettem részt, több blogcikkünkben is le tudtam írni a számlázás bizonyos funkcióit részletesen és a felhasználók számára érthetően, ahogyan azt az egyetemi oktatóimtól elsajátíthattam. Sok esetben pedig a hétköznapokban szerzett tapasztalataimat felhasználva jutottam előre a feladataim elvégzésében.

Az alkalmazásfejlesztést a C# nyelven keresztül mutattam be, mivel azt használtuk napi rendszerességgel. Összeszedtem ismereteim és tapasztalataim alapján a C# nyelv tulajdonságait és működési elveit. Természetesen nem csak a Back-End részt vettem alapul, a Billzone.eu egyedi alkalmazásfejlesztése cím alatt kitértem a HTML és a JavaScript tulajdonságait is. Ezekkel a nyelvekkel alkották meg a modern számlázó weblapot, aminek az összetettségéért a .NET keretrendszer felel, amit alkalmazunk. A programozási feladatokban legtöbbször hibakeresést és apróbb javításokat bíztak rám. Volt, amikor a kódban elhelyezett SQL utasítást kellett kiegészítenem, és volt, amikor csak meg kellett keresnem a hiba forrását, és azt továbbítottam a fejlesztő munkatársam számára, és ő oldotta meg a problémát.

A következő fejezetben a programok dokumentálásáról írtam. Ez egy fontos feladat a programozás folyamatában, ugyanis hiába írunk meg egy programot, ami sokak számára hasznos lehet, ha nem írjuk le a felhasználók számára a pontos működését. Úgy gondolom, hogy a munkám során majdnem a legtöbb időt a Billzone.eu dokumentálásával töltöttem. Ezek során frissítettem a fontosabb blogcikkeket, amiket a mindennapok során felkutatnak a felhasználók, mert a rendszer funkcióit mutatják be képek segítségével. Sok esetben a szöveget is át kellett fogalmaznom, mert a különböző funkciók azóta más elérési pontban találhatók meg, vagy éppen más elnevezést kaptak a frissítések során. Ezen kívül, minden hónapban én készítettem el a blogcikket az előző havi frissítésekről, amik elvégzésre kerültek általunk. Ha egy új funkció került bevezetésre, akkor arról egy külön cikket készítettem, hogy a felhasználók részletesebb betekintést nyerjenek az új funkcióba.

A záródolgozatom harmadik nagyobb fejezetében a tesztelést vettem alapul. A programok tesztelése legalább annyira fontos feladat, mint a program megírása, ugyanis tesztelés nélkül nem látjuk, ha nem megfelelően működik a programunk. Ez nem azt jelenti, hogy ha egyszer megírjuk, és hibátlanul működik a program az adott tesztesetek elvégzése során, hogy ha módosítást végzünk rajta, akkor nem változhat meg a működése. Minden javítás után érdemes több lépcsőben tesztelni, és ha lehetőségünk van rá, akkor több böngészőből, vagy több

operációs rendszerről. Ebbe beletartozik a mobiltelefonok és tabletek böngészőin való futtatás is. Ebben a fejezetben részleteztem a tesztelés 7 alapelvét és a tesztelő munkatárs feladatait. Úgy gondolom, hogy a munkahelyen tapasztaltak által több irányból is be tudtam mutatni a tesztelők feladatát, illetve fontosságát. Érdekes tapasztalat volt, hogy minden apró részletre mennyire oda kell figyelnünk, mert sokszor egy gomb nem működése miatt felborul az egész rendszer működése.

Az alkalmazásfejlesztés témát természetesen nem tudjuk egy keretbe zárni, ugyanis a számtalan programozási nyelv és technológia bemutatására nem elegendő ez a terjedelem. Véleményem szerint a Billzone.eu rendszeren keresztül egy komplex alkalmazásfejlesztési folyamatot tudtam bemutatni, amiben aktívan részt vettem a mindennapok során. A csoportmunka során megtapasztaltam, hogy minden külön munkafolyamatra milyen tudást igényelnek a feladat végrehajtásához a dolgozók, és rávilágítottak a projektvezetőim a pontos és rendszeres kommunikáció fontosságára.

7. Források

Dr Csetényi Arthur, Mohácsi László, Dr Várallyai László: Szoftverfejlesztés (2007)
(http://miau.gau.hu/avir/intranet/debrecen_hallgatoi/tananyagok/jegyzet/15-Szoftverfejlesztes.pdf)

Illés Zoltán: Programozás C# nyelven (2005)
(https://drive.google.com/file/d/0B8EJ1llCkLNCM0lkaG5SRlFTdVE/view)

https://www.fzolee.hu/fw/602_a_javascript_alapjai_valtozok

http://web.axelero.hu/fodorsi/html/htmlfuggelek.htm

http://web.axelero.hu/fodorsi/html/css1.html

http://trafo01.uw.hu/10F1_prelm/22_prelm.html

http://progalap.elte.hu/downloads/seged/eTananyag/lecke28_lap1.html

http://aries.ektf.hu/~gkusper/SzoftverTeszteles.pdf

https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_29_webfejlesztes_iii/lecke1_lap1.scorml#hiv6

https://www.billzone.eu/blog/2019/05/08/onszamlazas/

https://hu.wikipedia.org/wiki/Windows_Communication_Foundation

http://ade.web.elte.hu/wabp/lecke12_lap1.html

7.1 Képek