

SZAKDOLGOZAT



MISKOLCI EGYETEM

Automatizált szabadságnylvántartó rendszer tervezése és megvalósítása WEB-es környezetben.

Készítette:

Strutinszky Alex Roland
Programtervező informatikus

Témavezető:

Dr. Dudás László

MISKOLC, 2021

MISKOLCI EGYETEM

Gépészmérnöki és Informatikai Kar

Alkalmazott Matematikai Intézeti Tanszék

Szám:

SZAKDOLGOZAT FELADAT

Strutinszky Alex Roland (PXMQ3P) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: WEB-es alkalmazás, automatizálás, szabadság, nyilvántartás

A szakdolgozat címe: Automatizált szabadságnylvántartó rendszer tervezése és megvalósítása WEB-es környezetben.

A feladat részletezése:

- 1. Ismerje meg a Miskolci Egyetem Szabályzatának a munkahelyi szabadság kiadására vonatkozó pontjait!*
- 2. Készítse el a rendszertervét egy automatikus havi lejelentések megtételére képes éves szabadságnylvántartó rendszernek! A program vegye figyelembe a rendes és eseti szabadnapokat, tegye lehetővé a közalkalmazott szabadságválasztásában a maximális szabadságfokot, ugyanakkor tartsa be az összes vonatkozó országos és helyi szabályzat előírását. A rendszer webes felületen, kliens-szerver rendszerben működjön, kezelje az alkalmazotti, vezetői és rendszergazdai jogosultságokat. Nyújtson áttekintő felületet az alkalmazott saját szabadságainak állására, továbbá a vezető számára az összes alkalmazott vonatkozásában.*
- 3. Válassza meg az alkalmas programozási és futtatókörnyezetet a program elkészítéséhez, választását indokolja!*
- 4. Készítse el az alkalmazás programját és minden eshetőségre kiterjedő képzeletbeli adatokkal tesztelje!*
- 5. Készítse el az alkalmazás felhasználói kézikönyvét és ennek kivonatos ismertetésével mutassa be a használat lépéseit!*

Témavezető: Dr. Dudás László PhD (a műszaki tudomány kandidátusa, egyetemi docens)

A feladat kiadásának ideje: 2018. 09. 01.

.....
szakfelelős

EREDETISÉGI NYILATKOZAT

Alulírott **Strutinszky Alex Roland**; Neptun-kód: **PXMQ3P** a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírással igazolom, hogy *Automatizált szabadságnylvántartó rendszer tervezése és megvalósítása WEB-es környezetben.* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

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

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

Miskolc, év hó nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

.....

dátum

.....

témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):

konzulens (dátum, aláírás):

.....

.....

.....

.....

.....

.....

3. A szakdolgozat beadható:

.....

dátum

.....

témavezető(k)

4. A szakdolgozat szövegoldalt

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

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

.....

dátum

.....

témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve:

.....

dátum

.....

szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata:

a bíráló javaslata:

a szakdolgozat végleges eredménye:

Miskolc,

.....

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. Bevezetés	1
2. Magyar felsőoktatási szabadságnylvántartó szoftverszolgáltatás fejlesztése	3
2.1. Fogalmak, törvények, meghatározott feltételek	3
2.1.1. Szabadság	3
2.1.2. Fizetett szabadság	4
2.1.3. Alapszabadság	4
2.1.4. Pótszabadság	5
2.1.5. Közalkalmazottak szabadsága	5
2.1.6. Időarányos szabadság	5
2.1.7. Szabadságok felhasználása	7
2.2. Jelenleg elérhető megoldások	7
2.2.1. Nemzetközi szoftverek	7
2.2.2. Magyar fejlesztésű programok	8
2.3. Fejlesztési lehetőségek	9
2.4. Technológiai eszköztár	10
2.4.1. Verziókezelés és Verziókezelő rendszerek	10
2.4.2. Programozási nyelvek	11
2.4.3. Szoftver menedzsment eszköz	13
2.4.4. Keretrendszerek	13
2.4.5. Adatbázis, Verzió kezelt adatbázis, ORM	15
2.4.6. Szoftver architektúra	15
2.4.7. Tesztelés, Tiszta kód	16
2.4.8. Folyamatos integráció/Folyamatos szállítás (CI/CD)	17
2.5. A fejlesztéshez szükséges ismeretek összefoglalása	18
3. Automatizálható Webapplikáció Tervezése	19
3.1. Architektúra felépítése	19
3.1.1. Több rétegű alkalmazás	19
3.1.2. Megjelenítési réteg	20
3.1.3. Vezérlő réteg, és Adatközzetítő objektumok	22
3.1.4. Szolgáltatás réteg, és Belső objektumok	22
3.1.5. Adatbázis kezelő réteg, ORM, és Elemi objektumok	22
3.1.6. A hívási lánc összegzése	22
3.2. Adatbázis struktúra	24
3.2.1. Konfigurációs táblák	24
3.2.2. Felhasználói adatok	25

3.2.3.	Összetett adatok tárolása kisméretű táblákban	27
3.2.4.	Jóváhagyáshoz szükséges táblák	27
3.3.	Funkció tervezés	27
3.3.1.	Felhasználó és hozzáférés kezelés	28
3.3.2.	Szabadságolási szabályok konfigurálása	29
3.3.3.	Automatizált működés	30
3.3.4.	Szabadságolási statisztikák készítése	31
4.	Az alkalmazás megvalósítása Java Enterprise környezetben	32
4.1.	Mappastruktúra kialakítása	32
4.1.1.	Modularizált alkalmazás	32
4.2.	Összetett objektum előállítása JSON formátumból Lombok használatával	33
4.2.1.	Az ismétlődő programrészletek	33
4.2.2.	A Lombok működése	33
4.2.3.	Megváltoztathatatlan objektumok Lombokban	34
4.2.4.	JSON felépítésű String generálása Jacksonnal	34
4.3.	Automatizált Cron munkamenetek Springben	37
4.3.1.	A keretrendszer nyújtotta előnyök	37
4.3.2.	Cron munkamenet implementálása Spring-boot-ban	37
4.4.	Automatizált szabadság számolás tört év esetén, Nemzeti ünnepek tárolása	38
4.4.1.	Külső API hívása	38
4.4.2.	Mozgó ünnepek számolása	39
4.4.3.	Félautomata megoldás	39
4.5.	Folyamatos Integráció beállítása Gitlabon	39
4.5.1.	.gitlab-ci.yml konfiguráció	39
4.5.2.	További felhasználható funkció	40
5.	Rest applikáció tesztelése	41
5.1.	Unit tesztek	41
5.2.	Funkcionális tesztek Swagger UI segítségével	41
6.	Összefoglalás	43
6.1.	Fejlesztési lehetőségek	43
	Irodalomjegyzék	44

1. fejezet

Bevezetés

A 21. században az eddigieknél, sokkal nagyobb fejlődést mutató, egyre inkább digitalizálódó világban a különböző akár mindennapi adminisztrációk egyre nagyobb részét végezzük digitális felületeken. A papír alapú formanyomtatványokat már több helyen leváltották, a dinamikusan fejlődő és könnyedén változtatható WEB-es felületek. A dossziékat és irattartókat pedig adatbázisokra cserélték. Ezáltal új lehetőségek nyíltak a munkáltatók számára, hogy a bonyolult, időigényes, személyes ügyintézés igénylő műveleteket különböző szoftvereken keresztül végezzék el.

Mivel egy nagyobb szervezetben a munkavállalók száma akár a több százat is elérheti, és ezekre a személyekre meglehet, hogy külön szabályok vonatkoznak így szükségessé vált a szabadság nyilvántartó rendszerek bevezetése. Ezt az igényt természetesen hamar sikerült kielégíteni, de a piacon található szolgáltatások általános megoldást kínálnak amik a lehető legtöbb szervezeti egységre alkalmazhatóak és a legelterjedtebb szabályozásokat fedik le csupán.

Ezeket az igényeket eleinte jól megtervezett excel táblák és sablonok elégítették ki, de idővel ezek bonyolulttá és nehezen követhetővé váltak, valamint kimondottan sok helyet foglaltak. nem kellett sok idő, hogy a WEB applikációk megjelenésével és széles körben elterjedésével, több piac orientált rendszer szülessen.

A létező megoldások általánossága miatt, nem lehet specifikusan egy adott ország törvényeire alkalmazva konfigurálni, így a megannyi eltérő szabályozás miatt minden évben az alkalmazott számára elérhető szabadnapok kiszámolása, és jóváírása többnyire kézzel történik. Továbbá felmerül az a probléma is, hogy amennyiben egy alkalmazott-ra, vagy szervezeti egységre nem ugyanazok a szabályok vonatkoznak, abban az esetben milyen beállítások érhetőek el a különbségek kezelésére? Ezt a kérdést megválaszolva azt vesszük észre, hogy jelenleg nem elérhető olyan WEB-es szoftver ami alkalmazható vagy akár, automatizálható egy Magyar felsőoktatási intézményben, anélkül, hogy az eddigi papírmunkát maradéktalanul leváltsa. Ezáltal a felhasználók kénytelenek nehezen átlátható módon igényelni és a követni az elérhető és felhasznált szabadságaikat.

Az egyetemeken központosított nyilvántartást vezetnek az alkalmazottak szabadságairól, de ezzel kapcsolatban a következő kérdések merülnek fel: Vajon minden alkalmazottra egyforma jogszabályok érvényesek? A különböző jogszabályokat, képes külön kezelni a rendszer? Mennyire hozzáférhető ez az alkalmazottak számára? Nyomon tudják követni egyszerűen a felhasznált és hátralévő szabadnapjaikat az emberek? Mennyi időt vesz el tőlük egy ilyen ügyintézés? Van olyan lehetőség ami mindenkinek megfelel?

Ezen kérdéskört kutatva a dolgozatom célja, hogy tervezzek és megvalósítsak egy olyan WEB-es felületen elérhető alkalmazást, ami egy központi szerverre telepítve,

majd a megfelelő beállítások után, lehetővé teszi, hogy a Magyar szabályozásoknak megfelelően a felhasználók képesek legyenek gond nélkül naprakész információhoz jutni a szabadságaikkal kapcsolatban. Valamint a rendelkezésre álló adatok alapján a lehető legnagyobb mértékben automatizáljam a rendszert.

Dolgozatom során részletesen utánajártam az országunkban létező szabadság típusoknak, ezekhez kapcsolódó feltételeknek, valamint az átlagostól eltérő munkakörökre vonatkozó szabályozásoknak és implementáltam egy általam tervezett megoldást. Az implementáció során a próbáltam céges körülményeket szimulálni és a lehető legmodernebb technikai megoldásokkal dolgozni. Szakdolgozatom a bonyolult struktúra, az eltérő vagy akár változó statisztikai lehetőségek és az igényes megvalósítás időigényessége miatt nem tér ki a felsőoktatásban dolgozó közalkalmazottakra nem releváns pótszabadságokra valamint a szülési szabadság kimondottan bonyolult lehetséges kimeneteleire, csak és kizárólag az ideális esetre, továbbá a megvalósítás esetén: statisztikai funkciókra, a felettesi jóváhagyásra, valamint a felületre. Az felsorolt működési elemekre csak elméleti, rendszertervezés szintjén tér ki.

A kutatás és az implementáció remélhetőleg választ ad arra, hogy lehetséges-e megvalósítani teljesen automatizált szabadságnylvántartó applikációkat, valamint az általam megírt szoftver milyen további fejlesztésekkel, alkalmazható akár napi szinten a Magyar egyetemek tanszékein dolgozó oktatók, adminisztrátorok és egyéb munkakörökben dolgozók átláthatóbb szabadság menedzsmentjének érdekében.

2. fejezet

Magyar felsőoktatási szabadságnylvántartó szoftverszolgáltatás fejlesztése

Célszoftver fejlesztése esetén a szoftver fejlesztést megelőzi egy kutatás és igény felmérés, ami során feltérképezzük a felhasználói igényeket, és az ezekhez tartozó elérhető kínálatot. Az igényeknek megfelelően pedig új hasznos funkciókat adunk a saját programunkhoz.

A felhasználók tökéletes forrást biztosítanak az ötletek és igények kialakításához, viszont ezeknek a megértéséhez, megvalósításához szükség van háttér ismeretekre is mint például jogszabályok és alap fogalmak. Amik alapján megtervezhető a program struktúrája, és működése is jobban leírható.

Miután megvan a szükséges tudás, a célnak megfelelően technológiai eszköztárat kell választanunk, valamint tudnunk kell milyen fejlesztési metodológiát alkalmazunk a feladatok és eredmények nyomon követésére.

Ez a fejezet felvázolja, az elérhető lehetőségek hiányosságait a Magyar felsőoktatásban dolgozók mint felhasználók szempontjából, definiálja a témakörben használatos elméleti fogalmakat, valamint részletesen bemutatja a választott technológiai eszköztárat és folyamatokat.

2.1. Fogalmak, törvények, meghatározott feltételek

Fontos, hogy tisztában legyünk a tervezés során számunkra lényeges meghatározásokkal, szabályokkal. A fejezet ezen részében a tervezés szempontjából szükséges, a munkatörvénykönyvében definiált fogalmakat, szabályozásokat mutatom be.

2.1.1. Szabadság

A szabadság olyan napok összessége, melyet a munkavállaló szerződésében foglalt munkanap helyett, munkahelyétől és a munkavégzéstől távol tölt. A tervezett szabadnapokat, a munkáltató felé, a munkavállalónak meghatározott idővel előre kell jelezni. Mind a két félre vonatkozó törvények, ész szabályozások akár évente változhatnak így követünk kell az aktuális évben érvényes kiadásokat.

A szabadságnak két fajtája van:

- **Fizetett¹:** Szintén két részre bontható, munkanapnak minősül, szabályok alapján előre definiált, a munkavállaló pihenését és regenerálódását szolgálja.
- **Fizetés nélküli:** A fizetett szabadságon felül igénybe vett szabadnapok, alapos indokkal vagy közös megegyezés esetén élhet vele a munkavállaló, ezekre a napokra nem köteles a munkáltató fizetést biztosítani. [8]

2.1.2. Fizetett szabadság

A fizetett szabadság definíciója:

"A szabadságnak a munkavállaló pihenését, regenerálódását szolgáló rendeltetéséből következik, hogy az alapvetően a ténylegesen munkában töltött idő után illeti meg a munkavállalót minden naptári évben. Mindazonáltal a törvény tényleges munkavégzés hiányában is munkában töltött – így szabadságra jogosító – időnek minősíti a következő időtartamokat:

- a munkaidő-beosztás alapján történő munkavégzési kötelezettség alóli mentesülés időtartama (ilyennek minősülnek például a pihenőnapok, pihenőidők vagy a rendkívüli munkavégzés ellenértékéként biztosított szabadidő),
- a szabadság időtartama,
- a szülési szabadság időtartama,
- a gyermek gondozása céljából igénybe vett fizetés nélküli szabadság első hat hónapja,
- a naptári évenként harminc napot meg nem haladó keresőképtelenség (a naptári évre eső egyes keresőképtelenségek tartamát össze kell számítani),
- a tényleges önkéntes tartalékos katonai szolgálatteljesítés három hónapot meg nem haladó tartama,
- a munkavégzés alóli mentesülésnek az 55. § (1) bekezdés b)–k) pontban meghatározott tartama (az emberi reprodukciós eljárással összefüggő, egészségügyi intézményben történő kezelés tartama; a kötelező orvosi vizsgálata tartama; a véradáshoz szükséges, legalább négy órás időtartam; a szoptató anyát megillető, a szoptatás első hat hónapjában naponta kétszer egy, ikergyermekek esetén kétszer két órás, a kilencedik hónap végéig naponta egy, ikergyermekek esetén naponta két órás időtartam; hozzátartozó halálakor két munkanap;" [6]

A meghatározás, továbbiakban kitér az éves szabadság keret összetételére mely szerint minden munkavállaló részére "alap és pótszabadságból áll"[6].

2.1.3. Alapszabadság

Az alapszabadság, átlag munkavállaló esetén éves húsz napban van meghatározva. Ez bizonyos esetekben mint például, közalkalmazottak és oktatók eltérhetnek, de minden esetben meghaladják a korábban említett húsz napot.²[6] [10]

¹A dolgozat későbbi részében részletes bemutatásra kerül.

²Későbbi fejezetekben részletesebben tárgyalom ennek az eltérésnek a fontosságát.

2.1.4. Pótszabadság

A pótszabadság több különböző feltételnek megfelelően jól definiált és körülhatárolt értékek összessége, amelyek lehetnek állandó tehát érvénybelépésüket követően folyamatosan minden évben jogosult rá a munkavállaló. Valamint lehet eseti miszerint bizonyos életeseményekhez köthető, ideiglenes akár még korlátozott időtartamú felhasználhatóságról beszélhetünk.

A a 2.1 és a 2.2 ábrán láthatóak, a fajtái valamint az azokhoz tartozó munkanapban számolt értéke a 2013-as évben[6] [10].

Életkor után járó pótszabadság – először abban az évben illeti meg a munkavállalót, amikor a szükséges életkort betöltötte	
- a 25. életévtől	1 munkanap
- a 28. életévtől	2 munkanap
- a 31. életévtől	3 munkanap
- a 33. életévtől	4 munkanap
- a 35. életévtől	5 munkanap
- a 37. életévtől	6 munkanap
- a 39. életévtől	7 munkanap
- a 41. életévtől	8 munkanap
- a 43. életévtől	9 munkanap
- a 45. életévtől	10 munkanap
Gyermek ⁴ után járó pótszabadság – először a gyermek születésének évében, utoljára abban az évben, amikor 16. életévét betölti; mindkét szülőt megilleti	
- egy gyermek után	2 munkanap
- két gyermek után	4 munkanap
- kettőnél több gyermek után	7 munkanap

2.1. ábra. Pótszabadságok típusai és értékei

2.1.5. Közalkalmazottak szabadsága

A közalkalmazottak szabadságolása nagyban eltér az átlag munkavállaló szabadságolásától. Alap szabadságuk fizetési osztálytól függően változhat, valamint nem teljesen ugyan azok a pótszabadságokkal kapcsolatos szabályok vonatkoznak rájuk mint egy átlag munkavállalóra.

"A közalkalmazottat

- az „A”, „B”, „C” és „D” fizetési osztályban évi húsz munkanap,
- az „E”, „F”, „G”, „H”, „I”, „J” fizetési osztályban és a 79/C. §-ban említett munkakör betöltése esetén évi huszonegy munkanap

alapszabadság illeti meg." [25]

2.1.6. Időarányos szabadság

"A munkavállalót a szabadság naptári évenként (tárgyév) illeti meg, ezért a részére járó szabadságnapok számát minden tárgyévben meg kell állapítani. Az ennek megfelelően

- ha a munkavállaló gyermeke fogyatékos, fogyatékos gyermekenként	2 munkanap
Az apának gyermeke születése esetén járó pótszabadság ⁵ – legkésőbb a születést követő második hónap végéig, a munkavállaló kérésének megfelelő időpontban kell kiadni	5 munkanap
- ikergyermekek születése esetén	7 munkanap
A fiatal munkavállalónak járó pótszabadság – utoljára abban az évben jár, amikor a tizennyolcadik életévét betölti	5 munkanap
A föld alatt állandó jelleggel dolgozó munkavállaló pótszabadsága	5 munkanap
Az ionizáló sugárzásnak kitett munkahelyen naponta legalább három órát dolgozó munkavállaló pótszabadsága	5 munkanap
A rehabilitációs szakértői szerv által megállapítottan legalább ötven százalékos mértékű egészségkárosodással rendelkező munkavállaló pótszabadsága	5 munkanap

2.2. ábra. Pótszabadságok típusai és értékei

meghatározott szabadság a munkavállalónak, ha munkajogviszonya a tárgyévben egészében nem áll fenn, azaz a naptári év közben létesül vagy szűnik meg, arányosan jár. " [26]

Az éves szabadság megállapításakor tehát figyelembe kell vennünk, hogy a munkavállaló az adott évben ténylegesen mennyi munkanapot tölt alkalmazásban. Egy egyszerű matematikai képlet alkalmazásával számoljuk ki az alkalmazott éves szabadságát.

Egy példán keresztül bemutattva a következőképp számoljuk egy munkavállaló időarányos szabadnapjait egy évre.

"A munkavállalót alapszabadságként 20 munkanap szabadság illeti meg naptári évenként. Ezen túlmenően az Mt. számos jogcímen (életkor alapján, 16 éven aluli gyermekekre tekintettel, speciális munkakörökben stb.) határoz meg pótszabadságokat, amelyek együttesen is megilletik a munkavállalókat.

Az időarányosítás elvégzéséhez első lépésben meg kell állapítani a munkavállalót megillető szabadság mértékét. Ezt attól függetlenül munkanapban kell megtenni, hogy a munkáltató a szabadságot egyébként munkanapban vagy órában tartja nyilván. Így például, egy harminchárom éves, két tizenhat év alatti gyermeket nevelő munkavállalót, aki állandó jelleggel föld alatt dolgozik, az alábbi jogcímenek illeti meg szabadság:

- alapszabadság (20 munkanap)
- életkori pótszabadság (4 munkanap)
- gyermekek után járó pótszabadság (4 munkanap)
- föld alatti munkavégzés miatt (5 munkanap).
- föld alatti munkavégzés miatt (5 munkanap).

Azaz, a munkavállalónak mindösszesen 33 munkanap szabadsága van.

Az így megállapított összes szabadságot arányosítani kell, a munkaviszony év közben történő keletkezése, illetve megszűnése esetén. A fenti példában szereplő munkavállalót, amennyiben munkaviszonya 2016. július 1-jén fog elkezdődni (ekkor tárgyévben legfeljebb 184 napot fog munkaviszonyban állni) $33/365 * 184 = 16,64$ munkanap szabadság illetné meg.

Tekintettel arra, hogy az így kiszámított szabadság töredéknapot eredményez, alkalmazni kell az Mt. azon szabályát, hogy a fél napot elérő töredéknapi egész napnak számít. Azaz, a munkavállalót mindösszesen 17 munkanap szabadság illeti meg 2016-ban." [7]

Tehát az általános egyenlet a következő:

$$\frac{MunkavallaloEvesSzabadsaga}{TargyEvNapokban * TargyEvbenHatralevoMunkanapok} = \lceil szabadsag \rceil \quad (2.1)$$

2.1.7. Szabadságok felhasználása

Fontos tudni, hogy a munkavállaló nem rendelkezik teljes mértékben a számára meghatározott éves szabadság kerettel.

"A munkáltató évente hét munkanap szabadságot legfeljebb két részletben a munkavállaló kérésének megfelelő időpontban köteles kiadni azzal, hogy a munkavállalónak erre vonatkozó igényét legalább tizenöt nappal a szabadság kezdete előtt be kell jelentenie. A hét napos szabály jelentősége, hogy amennyiben a munkavállaló szabályszerűen bejelentette szabadságigényét, azt a munkáltató nem mérlegelheti.

Fontos, hogy ez a hét napos szabály a munkaviszony első három hónapjában nem köti a munkáltatót, azaz a munkaviszony első három hónapjában a munkavállaló a hét szabadnap általa meghatározott időpontban történő felhasználási jogával nem élhet. Ez utóbbi szabályt a legtöbbször – tévesen – úgy értelmezik, hogy próbaidő alatt, illetve az első három hónapban nem lehet szabadságra menni. [...]

Fontos kötelezettsége a munkáltatónak, hogy a szabadságot úgy kell kiadnia, hogy a munkavállaló naptári évenként egy alkalommal, legalább tizennégy egybefüggő napra mentesüljön a munkavégzési és rendelkezésre állási kötelezettsége alól. Ettől a szabálytól azonban a felek eltérhetnek.

A tizennégy napos szabály tekintetében – a szabadságként kiadott napon túl – a heti pihenőnap (heti pihenőidő), a munkaszüneti nap és az egyenlőtlen munkaidő-beosztás szerinti szabadnap vehető figyelembe." [28]

2.2. Jelenleg elérhető megoldások

A rengeteg szabályozásból, csak párat kiemelve jól látszik, hogy az automatizáláshoz szükséges számítások nem túl bonyolultak. A törvények összetett struktúrája, viszont országoként eltér, sőt a munkavállalókra különböző jogszabályok vonatkoznak amennyiben azok közalkalmazottak. Ezen információk fényében a vizsgálható, az elérhető megoldások amiket a jelenlegi piac kínál.

2.2.1. Nemzetközi szoftverek

Szinte számtalan applikáció elérhető a nemzetközi piacon, és ezek mindegyike minőségi szolgáltatás nyújt egy modern és könnyen kezelhető design keretében. Összességében

mindegyikről elmondható, hogy ezek a szolgáltatások havidíjas előfizetéssel, működnek és többnyire tartozik hozzájuk egy profi támogatói csoport is akik mind konfigurálás, mind felhasználói problémák esetén is elérhetőek.

Továbbá az is jól látszik, hogy több országban jelen vannak így nem csak az általános pótszabadságokat tudjuk ezeken keresztül kezelni, de akár saját személyre szabott típust is definiálhatunk.

Hátrányuk, viszont nem csak a havi díjban megszabott ár hanem az ehhez tartozó felhasználói létszám keret vagy akár a csomagban definiált funkció korlátozás is. Látható még, hogy ezek az alkalmazások vagy csak egy funkcióként kezelik a szabadság nyilvántartást, vagy egy több szolgáltatásból álló egység külön része ami arra enged következtetni, hogy bár külön álló programként is funkcionál, bizonyos előnyök csak a többi testvér szolgáltatással együtt használhatóak.

Végül láthatjuk, hogy egyik alkalmazás bemutató anyaga sem tér ki a közalkalmazottakra, így azt feltételezhetjük, hogy ezek a szolgáltatások nincsenek felkészítve a közalkalmazotti különbségekre.

A 2.1 Nemzetközi programok összehasonlítása táblázatban látható néhány elérhető megoldás.

2.1. táblázat. Nemzetközi programok összehasonlítása.

Termék megnevezése	Tulajdonságok	Csomag árak
freshworks, freshteam, Time Off[11]	Teljes körű HR szolgáltatás, felhasználó adatbázis, munkaidő követés, alapvető vagy akár bonyolult szabadság kezelés	Ingyenes(nem tartalmaz egyéni szabadság kezelési lehetőséget), 75\$ - 250\$/hó/50 felhasználó
Leave Dates[5]	Idő menedzsment, szabadság kezelés	Ingyenes 5 felhasználóig, 1\$/hó
AnnualLeave[4]	Idő menedzsment, szabadság kezelés	Árajánlatérés alapján, elérhető kalkulátoruk szerint 22\$/hó(20 főre)
Calamari[29]	Szabadság kezelés, felső vezetői jóváhagyási rendszer, nemzetközi naptárak	Felhasználók mennyiségétől függ 25\$/hó(20 főre)

2.2.2. Magyar fejlesztésű programok

A piacon található Magyar eredetű megoldások többsége valamiért még mindig az összetett excel táblázatokon alapul, amikben bonyolult függvények és macro programozással valósítják meg a nyomon követést.

Ezeknek a megoldásoknak a legnagyobb hibája a nehezen megoldható konkurens kezelés valamint, a felhasználó számítógépével való szoros függőség. Mindezek mellett sérülékeny, az óvatlan szerkesztés és mentés esetén könnyen az egész táblázat hamis adatokat mutathat. Esetleges hiba bekövetkeztével a pótlásuk nehézkes, szinte lehetetlen.

Fellelhetők itt is kifejezetten a témával kapcsolatos szoftverek vagy akár teljes szolgáltatások is. Ezek a lehetőségek többnyire komplex rendszerek több, esetenként akár felesleges funkcióval.

A a 2.2 Magyar megoldások összehasonlítása táblázatban látható néhány elérhető megoldás.

2.2. táblázat. Magyar megoldások összehasonlítása

Termék megnevezése	Tulajdonságok	Csomag árak
Szabadság tervező és nyilvántartó[30]	Szabadság nyilvántartó excel táblázat	6 felhasználóig ingyenes, 21-50 felhasználóig 21.000 Ft + ÁFA/hó
excelmester[9]	Szabadság nyilvántartó excel táblázat	Ingyenes
HRmaster[23]	Összetett erőforrás menedzser szoftver	Árajánlat kérés alapján
PREDOR[24]	Munkaidő nyilvántartó és biztonsági rendszer	Árajánlat kérés alapján

2.3. Fejlesztési lehetőségek

Az elérhető megoldásokat összegezve láthatjuk, hogy többségük a munkavállaló szempontjából könnyen használható, mégis számunkra kényelmetlen konfigurációs problémákat vethet fel, amennyiben alkalmazottainkra eltérő szabályokat kell alkalmaznunk. Megfigyelhetjük azt is, hogy az összetettebb statisztikai lehetőségeket biztosító alkalmazások meglehetősen költségesek egy felsőoktatási intézmény számára, és akár megannyi számunkra kevésbé hasznos később kihasználatlan funkciót is tartalmazhatnak.

Továbbá általánosan elmondható a promóciós anyagok alapján, hogy az automatizált működés nincs megvalósítva.

Amennyiben leszűkítjük a felhasználók körét a felsőoktatásban dolgozó oktatókra és az azokkal együtt, tevékenykedő adminisztrátorokra láthatóvá válik, hogy bár az egyetemek használnak központi alkalmazásokat de egy-egy intézetben már megjelennek az eltérő szabályok. Egy egységben egyszerre vannak jelen normál, és közalkalmazotti jogviszonnyal rendelkező munkavállalók is, ezen felül szervezeti egységenként az az tanszékenként, intézetenként a szabadságok jóváhagyásáért az adott egység vezetője a felelős. Felelősségi körébe viszont beletartozik egy éves szabadságolási terv előállítása is amelynek be kell tartania a szabadság felhasználásával kapcsolatos jogszabályokat is. Az említett tervezetet valamint időközi statisztikákat jelenleg manuálisan állítják össze és továbbítják a központi rendszerbe.

Mindent egybevetve a piacon fellelhető applikációk több extra funkciótól mentes de mégis számunkra specifikus alkalmazást hozhatunk létre a következő működések megvalósításával:

- A felhasználó egyedi szabadságolással kapcsolatos adatait tartalmazó adatbázis
- A Magyar törvényekre specifikus szabadság típusoknak megfelelő konfigurálási lehetőség
- Dinamikus alkalmazhatóság, eltérő alkalmazotti jogkörök esetén.
- Automatikus kalkuláció teljes és részleges tárgy év esetén.
- Felettesi jóváhagyás támogatása.
- Éves szabadságolási tervező / Automatikus generálás.
- Tetszőleges statisztikák lekérdezése.
- Felhasználó jól átlátható, követhető szabadnap menedzsmentje.

2.4. Technológiai eszköztár

A konkrét tervezési lépések megkezdése előtt rendkívül fontos az alkalmazásunk számára megfelelő technológiai háttérrel biztosítani.

Választásunk során nem elég figyelembe venni mi az amit kezelni tudunk, fontos előretekintően lehetőleg minél modernebb technológiákat alkalmazni amik nagyban könnyíthetik bizonyos műveletek elvégzését a megvalósítás során, valamint lehetőséget nyújtanak a későbbi fejlesztés egyszerűsítésére is.

Ellenben nem érdemes kiforratlan kezdetleges de divatos eszközökhöz sem, hozzátűközni magunkat!

2.4.1. Verziókezelés és Verziókezelő rendszerek

"Verziókezelés alatt több verzióval rendelkező adatok kezelését értjük. Leggyakrabban a mérnöki tudományokban és a szoftverfejlesztésben használnak verziókezelő rendszereket fejlesztés alatt álló dokumentumok, tervek, forráskódok és egyéb olyan adatok verzióinak kezelésére, amelyeken több ember dolgozik egyidejűleg. Az egyes változtatásokat verziószámokkal vagy verzióbetűkkel követik nyomon." [22]

Lényeges pontja a fejlesztésnek, hogy a forráskód, ne csak a saját gépünkön legyen elérhető, amennyiben csapatban dolgozunk ez a szempont kiemelten fontos lehet, nagyban könnyítheti a mindennapokat, ha egymástól függetlenül könnyen tudunk változtatásokat eszközölni a kódbázison.

Ezeknek a rendszereknek a gyakorlati haszna túlmutat a forráskód konkurens kezelésén, stabil verziók esetén szerves részét képezik mindennapi használatban lévő úgynevezett "éles" vagy "production" környezetbe való kitelepítését.

SVN "A Subversion (SVN) egy verziókezelő rendszer, melyet a CollabNet Inc. indított 2000-ben. Fájlok aktuális verzióinak és történeteinek kezelésére használják, mint például forráskódok, weboldalak és dokumentációk. A célja, hogy a legkompatibilisebb utódja legyen a széles körben használt Concurrent Versions System (CVS)-nek." [14]

Az SVN alapú rendszerek jól használhatóak a lineáris fejlesztésben, de agilis fejlesztési modell esetén sokkal nem elég rugalmasak és nehezebben kezelhetők.

Korábban a legnépszerűbb használatban lévő rendszer, mai napig megtalálható a piacon, viszont a modernebb koncepciók elterjedésével egyre ritkábban találkozhatunk SVN-ben kezelt kódbázissal.

Git "A Git egy nyílt forráskódú, elosztott verziókezelő szoftver, vagy másképpen egy szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt. A Gitet eredetileg Linus Torvalds fejlesztette ki a Linux kernel fejlesztéséhez. Minden Git munkamódszert egy teljes értékű repository teljes verziótörténettel és teljes revíziókövetési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől." [16]

Gyors és skálázható rendszer, könnyen és dinamikusan kezelhető kis és nagy létszámú csapatok esetén is. Gyorsan tanulható és könnyen átlátható, a lokális repository már kezdő fejlesztőként is segíti és gyorsítja az egyes fejlesztési műveleteket. Megfelelő beállítások, használat esetén a központi repository stabil ága jól védett, ezáltal folyamatos lehetőséget biztosít az általunk fejlesztett szoftver legfrissebb működő állapotának az "éles" végfelhasználók számára elérhető környezetre való kitelepítését.

Jelenleg széles-körben használt, elterjedt és népszerű technológia. A megvalósítás során én is ezt a technológiát használtam fel.

2.4.2. Programozási nyelvek

Webfejlesztés esetén több meghatározó területet definiálhatunk, de ezek közül is leginkább a Backend az az háttér rendszeri fejlesztés, valamint a Frontend tehát a felhasználói felület és megjelenítés fejlesztés amit főbb tagolásnak nevezhetünk. Ezek a területek funkciójukban és feladatkörükben elválaszthatóak, többnyire külön micro alkalmazásokként működnek, így fontos megemlíteni, hogy bár minden esetben egymással kommunikálnak, többnyire egymástól független programozási nyelveken íródnak.

Mind a két területről elmondható, hogy nagy választékkal rendelkeznek nyelvek területén, ezek folyamatosan megújuló megoldások ezért ezen a területen kiemelten fontos, hogy körültekintően válasszunk! Egy friss és dinamikus megoldásokat kínáló nyelv gyakran lehet kiforrotlan és kevésbé előnyös a későbbi fejlesztések során a jól bevált már ismert nyelvekkel szemben.

Háttérrendszer, Backend Ezen a területen bár mostanában kezd elterjedni a node.js, mégis a C# és Java nyelveket mondhatjuk meghatározónak. Bár mind a C# is elismert és több helyen használt, mégis mondhatni jelenleg a Webfejlesztés piacán a Java az elterjedtebb és támogatottabb nyelv.

Java "A Java általános célú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a '90-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle.

A Java alkalmazásokat jellemzően bájtkód formátumra alakítják, de közvetlenül natív (gépi) kód is készíthető Java forráskódból. A bájtkód futtatása a Java virtuális géppel történik, ami vagy interpretálja a bájtkódot, vagy natív gépi kódot készít belőle, és azt futtatja az adott operációs rendszeren. Létezik közvetlenül Java bájtkódot futtató hardver is, az úgynevezett Java processzor.

A Java nyelv a szintaxisát főleg a C és a C++ nyelvektől örökölte, viszont sokkal egyszerűbb objektummodellel rendelkezik, mint a C++. A JavaScript szintaxisa és neve

hasznoló ugyan a Java-hoz, de a két nyelv nem áll olyan szoros rokonságban, mint azt ezekből a hasonlóságokból gondolhatnánk. [...]

A nyelv első tulajdonsága, az objektumorientáltság („OO”), a programozási stílusra és a nyelv struktúrájára utal. Az OO fontos szempontja, hogy a szoftvert „dolgok” (objektumok) alapján csoportosítja, nem az elvégzett feladatok a fő szempont. Ennek alapja, hogy az előbbi sokkal kevesebbet változik, mint az utóbbi, így az objektumok (az adatokat tartalmazó entitások) jobb alapot biztosítanak egy szoftverrendszer megtervezéséhez. A cél az volt, hogy nagy fejlesztési projekteket könnyebben lehessen kezelni, így csökken az elhibázott projektek száma. [...]

A Java legfontosabb része a Java virtuális gép (Java Virtual Machine – JVM). A JVM mindenütt jelen van (szinte mindenféle berendezés, chip és szoftvercsomag tartalmazza), így a nyelv középszintként és platformként egyaránt működik. Ugyanakkor a nyelv „platformfüggetlen” is, mert a Java virtuális gépek interpretálják a szabványos Java bájtkódot. Ez azt jelenti, hogy egy PC-n megírt Java program minimális módosítás után ugyanúgy fog futni egy javás telefonon is. Innen jön az írd meg egyszer, futtasd bárhol kifejezés. Ez jelentős költségcsökkenést eredményez, mert a kódot csak egyszer kell megírni.”[19]

A megvalósítás során Java 11-es verziót használtam, melyben elérhetőek a lambda és stream nyelvi elemek.

Megjelenítés, Frontend A megjelenítési réteg webapplikációk esetén többnyire három fő részből áll. A HTML leírónyelven íródó struktúra adja az oldalak felépítését, vázát. Ezt kiegészíti egy másik leíró nyelv a CSS ami minden esetben a megjelenő designért felelős. Valamint ide tartoznak még a különböző script nyelvek mint a javascript amelyek az böngészőben futó logikáért felelősek.

Java nyelv esetén beszélhetünk még a JSP technológiáról ami az említett frontend technológiákat ötvözte valamint lehetővé tette, hogy Java kódot építsünk be HTML-be. Ezzel elértük, hogy az alkalmazásunk szerves része legyen egy általunk generált megjelenés is.

A mára elavultnak számító JSP technológiát egyszerűbb esetekben a hasonló működési elven alapuló szerver oldalon generált megjelenést biztosító Java template engine-ek váltották fel több helyen. Használatukban igen hasonlóak az elődjükhöz, viszont a különböző Java logikák mint az if vagy a for használata sokkal egyszerűbb és átláthatóbb. Az egyik legnépszerűbb alkalmazás ezen a téren a Thymleaf.

A legmodernebb megoldás jelenleg az Angular logikával megvalósított megjelenítés. Ezen esetben is jelen van a HTML, valamint a CSS viszont az oldalunk tartalmát és akár megjelenítését is tudjuk változtatni akár csak a sima javascript esetén. Ez az elképzelés viszont objectum orientált alapokra helyezi a frontend fejlesztést, valamint saját különálló programként tekint a megjelenítési réteg kódbázisára.

Tehát ebben az esetben arról beszélhetünk, hogy önállóan is futtatható a programunk megjelenítési rétege, így az a háttérrendszerrel elkülönítve fejleszthető a két alkalmazás. Csupán kommunikációs kapcsolat van köztük így amennyiben arra igény van a megjelenítés, valamint a szerver oldali logika egymástól függetlenül cserélhető is.

Itt érdemes még megemlíteni az előre definiált designt biztosító Bootstrap szolgáltatást is, ami lehetővé teszi, hogy a megjelenítést gyorsan felépítsük már létező CSS

elemekkel.

2.4.3. Szoftver menedzsment eszköz

Mivel meglehetősen időigényes akár a mindennapi fejlesztés terén az alkalmazásunk futtatására különféle parancsok megfelelő sorozatát kiadni, így igény keletkezett az automatizálható szoftver menedzsment eszközökre. Az egyik ilyen legelterjedtebb az Apache Ant volt eleinte. Viszont mára sokkal ismertebb és elterjedtebb a szintén Apache fejlesztésű Maven használata.

Maven "Az Apache Maven (röviden Maven) egy szoftver, amelyet szoftverprojektek menedzselésére és a build folyamat automatizálására lehet használni. Jason van Zyl készítette 2002-ben. Funkcionalitásában hasonlít az Apache Ant eszközhöz (és némi hasonlóságot mutat a PHP-s PEAR-rel és a perles CPAN-nal, de egyszerűbb és XML-alapú a konfigurációs modellje). A projektet az Apache Software Foundation hosztolja, ahol korábban a Jakarta Projekt részeként működött.

A Maven bevezeti a POM, azaz a Projekt Objektummodell (angolul: Project Object Model) fogalmát. Egy POM egy buildelendő projektet ír le és annak függőségeit. Az egyes lépéseket céloknak, angolul goal-oknak nevezik. Vannak előre definiált célok a tipikus feladatokra, mint például a kód fordítása és csomagolása, de a felhasználónak lehetősége van saját célokat is definiálni a projektspecifikus lépések végrehajtására.

A Maven hálózatképes, tehát szükség esetén dinamikusan is le tud tölteni komponenseket. Repository névvel illetik a különböző hosztok fájlrendszereinek azon mappáit, ahol a letölthető komponensek találhatók. A Maven nem csak a repository-kból való letöltést támogatja, hanem a készült szoftvercsomag feltöltését is. Ezzel az automatizálható le- és feltöltési mechanizmussal a Maven de facto szabványt próbál teremteni, de elég lassan fogadja el a Java közösség.

A Maven plugin alapú architektúrája lehetővé teszi tetszőleges parancssorból vezérelhető alkalmazás használatát. Ez elméletileg lehetővé teszi tetszőleges programnyelvekhez való pluginek készítését, de a gyakorlatban minimális mennyiségű nem javás plugin készült."[18]

Fontos megemlíteni, hogy bár a Maven jelenleg a legelterjedtebb manapság egyre népszerűbb a Gradle is, ami átláthatóbb és dinamikusabban konfigurálható, mégis egyetlen hátránya van ami miatt kevesebben használják és ez csupán az, hogy kevésbé ismert.

2.4.4. Keretrendszerek

Habár csak és kizárólag Java nyelv használatával is képesek vagyunk megvalósítani egy számunkra megfelelő alkalmazást, mégis érdemes megkönnyíteni a saját dolgunkat különböző keretrendszerek használatával. Korábban ilyen volt a Java EJB, mára viszont sokkal kifinomultabb megoldások léteznek, a webalkalmazások terén az egyik legismertebb a Spring valamint az arra épülő Spring-boot.

Spring "A Spring egy nyílt forráskódú, inversion of controlt megvalósító Java alkalmazás keretrendszer. A Spring keretrendszer több önálló modulból épül fel, amelyek az alábbi szolgáltatásokat nyújtják a fejlesztők számára:

- Inversion of control konténer: a Java objektumok életciklusának kezelése és az alkalmazás-komponensek testreszabása.
- Aspektus orientált programozási paradigma követésének lehetősége.
- Adatelérés: lehetőség van relációs adatbázis-kezelő rendszerek JDBC segítségével történő elérésre, és objektum-relációs lekérzések, NoSQL integrálására.
- Tranzakciókezelés: többféle tranzakció kezelő API-t tartalmaz.
- Modell-nézet-vezérlő szabvány: egy HTTP- és servlet alapú keretrendszer segítségével valósítható meg, amelyet arra fejlesztettek ki, hogy bővíthetők és személyre szabhatóak legyenek a webszolgáltatások
- Távoli eljáráshívás kezelő keretrendszer: biztosítja a RPC alapú, hálózaton keresztül történő Java objektum importokat és exportokat. További támogatást nyújt a RMI, a CORBA és HTTP alapú protokollok
- használatára, beleértve a webszolgáltatásokat (SOAP) is.
- Kötegelési eljárás támogatása.
- Azonosítás és azonosságkezelés: biztonsági folyamatok konfigurálása, melyet a Spring projekthez tartozó, Spring Security alprojekt tesz lehetővé a különféle protokollok és módszerek biztosításával.
- Üzenetkezelés: a JMS API-n keresztül történő általános üzenetkezelés továbbfejlesztése érhető el.
- Tesztelés: segítséget nyújt a unit- és az integrációs teszt írására."[17]

Inversion of control (IoC) avagy a kontroll megfordítása "főleg objektum-orientált programozási nyelvekben használt technika a komponensek összeillesztésére, konfigurálására és kezelésére.

A technika lényege, hogy a komponenskezelést (pl. létrehozást, példányosítást, paraméterezést, megszüntetést, metódus hívás) kiemeljük a programkódból, és általában egy külső keretrendszerre bízuk, mint pl. a Spring.

Célja a modularitás növelése és bővíthetővé tétele. Az objektumorientáltság nem feltétel. A kifejezést Robert C. Martin és Martin Fowler népszerűsítette.

Egy rokon minta a függőség befecskendezése, ami a függőségeket megosztott absztrakcióval kezeli a felső és az alsó rétegek között. Kapcsolódik az eseményvezérelt programozáshoz is, mert azt gyakran a kontroll megfordításával valósítják meg. A felhasználó kódja csak az eseményeket dolgozza fel, míg az eseményciklus és az események, üzenetek közvetítése a keretrendszerre vagy a futtató környezetre van bízva."[20]

Spring konfigurálása Ahhoz, hogy az általunk megírt osztály a keretrendszerben létezzen, konfigurálnunk kell belőle egy példányt amit fel tud dolgozni és a saját kontextusába be tud illeszteni. Ennek az úgynevezett bean definíciónak három módja van. XML, Java valamint annotáció alapú konfigurációból választhatunk, de akár keverhetjük is őket ami nehezíti a kód megértését így semmiféleképp nem ajánlott. Mindegyiknek megvan természetesen a maga előnye és hátránya. Csak rajtunk múlik

melyiket használjuk de fontos, hogy lehetőleg maradjunk egy félénél és legyen konzisztens a kódunk.

Spring-boot Egy natív Spring alapú szolgáltatás melynek lényege, hogy gyorsan és még egyszerűbben tudjunk önálló webalkalmazásokat létrehozni. Amíg az egyszerű Spring esetén az összes elemet nekünk kell összerakni és bekonfigurálni, a boot esetén előre definiált egységeket használunk ami egy kezdetleges konfigurációval rögtön futtatható programot biztosít számunkra.

Továbbá ezen külső könyvtárak és megvalósítások újra konfigurálása is sokkal egyszerűbb.

Tartalmaz előre meghatározott annotációkat amik használatával másodpercek alatt hozhatunk létre a keretrendszer kontextusában létező osztályokat.

2.4.5. Adatbázis, Verzió kezelt adatbázis, ORM

Napjainkban egyre több típusú adatbázis létezik, és mindegyiknek megvan a leghasznosabb felhasználási területe, már nem csak a jól ismert SQL típusok érhetőek el de megjelent a NoSQL valamint sok más koncepció alapján működő lehetőség is.

A megvalósítás szempontjából szinte tökéletesen megfelel a mindenki által ismert SQL típus, ez egy igen széles-körben elterjedt és alkalmazott adatbázis.

Liquibase "egy ingyenes, nyílt forráskódú, adatbázisfüggetlen változáskövető eszköz, ami jól használható az elterjedt verziókövető eszközökkel. Lényege, hogy a szükséges adatbázismódosításokat nem közvetlenül az adatbázison hajtjuk végre, hanem minden változást struktúrált módon, egy ún. changelog fájlban tartunk nyilván. Ezt a changelog fájlt aztán a Liquibase parancssoros alkalmazás segítségével vagy kódból, a Liquibase Java API-val tudjuk érvényesíteni. Ennek használata a következő előnyökkel jár:

- a changelog fájl feltölthető verziókövetőbe, ezáltal megvalósíthatjuk adatbázisunk verziókövetését;
- a Liquibase számon tartja, hogy mely változások kerültek korábban már érvényesítésre adatbázisunkon, és csak a szükségeseket futtatja." [3]

Objektum-relációs leképezés(ORM) "Az objektum-relációs leképezés (angolul Object-Relational Mapping) egy programozási technika adatok konvertálására nem kompatibilis típusos rendszerek és objektumorientált programozási nyelvek között. Lényegében egy „virtuális objektum-adatbázist” hoz létre, amit a programozási nyelvben használhatunk." [13]

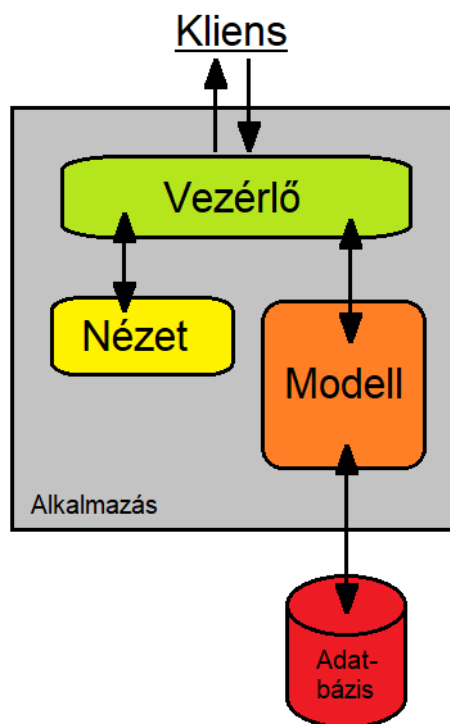
Egyik legismertebb megvalósítása a JPA, ami nem csak a részét képezi a spring-bootnak, hanem saját ORM rendszert épít rá Spring Data néven.

2.4.6. Szoftver architektúra

"A modell-nézet-vezérlő (MNV) (angolul model-view-controller) a szoftvertervezésben használatos programtervezési minta. Összetett, sok adatot a felhasználó elé táró számítógépes alkalmazásokban gyakori fejlesztői kíváncsi az adathoz (modell) és a felhasználói felülethez (nézet) tartozó dolgok szétválasztása, hogy a felhasználói felület ne

befolyásolja az adatkezelést, és az adatok átszervezhetőek legyenek a felhasználói felület változtatása nélkül. A modell-nézet-vezérlő ezt úgy éri el, hogy elkülöníti az adatok elérését és az üzleti logikát az adatok megjelenítésétől és a felhasználói interakciótól egy közbülső összetevő, a vezérlő bevezetésével.

Hagyományosan asztali felhasználói felületekhez használt, de manapság már web-alkalmazásokhoz is népszerűvé vált. Népszerű programozási nyelvek mint a JavaScript, Python, Ruby, PHP, Java, C# és Swift már külön telepítés szükségessége nélkül rendelkeznek MNV keretrendszerekkel web- és mobilalkalmazások fejlesztésére." [21]



2.3. ábra. MNV reprezentációja

Az MNV egy összetettebb változata a több szintű architektúra amiben a korábban említett három rétegen felül saját rétegeket definiálunk és azok között belső transzformációkkal továbbítjuk az adatokat. Ilyen további rétegek lehetnek az Adat továbbító-réteg(DTO), a tartomány réteg (Domain) valamint az adat hozzáférési réteg (dal).

A többszintű alkalmazás architektúrák könnyebben átalakíthatóak Microservice architektúrákká, melyek fejlesztése és üzemeltetése egyszerűbb, költséghatékonyabb és stabilabb.

2.4.7. Tesztelés, Tiszta kód

A fejlesztési folyamatok során elengedhetetlen, hogy folyamatosan kipróbáljuk és ellenőrizzük az általunk írt logikát. Ezeket az ellenőrzési folyamatokat nevezzük tesztelésnek.

A felületen történő funkcionális tesztől egészen a legapróbb kódsorig rengeteg lehetőségünk van a programunkat különböző tesztek alá vetni, hogy ténylegesen az elvárt működésnek megfelelően viselkedjen a szoftverünk. Ezekhez támogatást nyújt számunkra a már említett Maven, valamint a Spring keretrendszer is.

Különböző tesztelést segítő megoldások léteznek ilyen például a JUnit4, JUnit5 valamint a TestNG is, hogy megtudjuk határozni a tesztek hatáskörét és garantáljuk egymástól való függetlenségüket, valamint, hogy meghatározzuk külső függőségek visszatérési értékét objectum utánzó objectumokat más néven Mock-okat használunk.

Egységtesztelés "A számítógép-programozásban az egységtesztelés a szoftvertesztelésnek egy olyan módszere, amelynek során a forráskód egységeit (egy vagy több számítógépes program modul készletet) a kapcsolódó vezérlő adatokkal, a felhasználási-és a működtető eljárásokkal együtt tesztelik annak meghatározására, hogy azok elérik-e kitűzött céljukat." [15]

Egység tesztelés során fontos meghatároznunk, mi az a legkisebb egység amit tesztelünk, általános megállapodás szokott lenni, hogy egy függvényt tekintünk tesztelendő egységnek. További megállapodás, hogy csak publikus elérésű függvényekre írunk tesztet, viszont a vélemények sokszor eltérnek a tesztesetek számát illetően. Vannak nézetek ami szerint minden lehetséges esetet le kell fedni, de a gyakorlatban legtöbbször csak az elvárt működést valamint a logika különböző agait szokás lefedni.

Manuális tesztelés Ahhoz, hogy az elvárt működést teljesen le tudjuk ellenőrizni, működés közben teszt adatok segítségével úgymond felhasználóként kipróbáljuk a funkciókat a felületen keresztül. Amennyiben csak háttérrendszerrel rendelkezünk ez a tesztelés továbbra is alkalmazható, a korábban már tárgyalt MNV modell miatt. A vezérlőt közvetlenül meg tudjuk hívni szimulálva ezzel a megjelenítési rétegtől érkező kérést, a szerver oldali alkalmazás pedig kiszolgálja azt egy JSON válasszal.

Ennek vizualizációjára szolgál a swagger-UI vagy más néven OpenAPI eszköz.³

Tiszta kód Fontos megemlíteni, hogy a tesztek olvashatósága és egyszerűsége érdekében jól tagolt olvasható kódot kell produkálnia a fejlesztőnek. Ehhez szükség van előre definiált mérőszámokra mint például sorok hossza, függvények sorszáma, ciklo-matikus komplexitás.

A nehezen olvasható és tesztelhető kódok, nagyobb valószínűséggel tartalmazznak hibát és szinte biztos, hogy újrírásra szorulnak.

A tiszta kód szabályai mindig a fejlesztőcsoport megegyezése alapján történnek, a szabályok betartása viszont nagyban könnyíti a későbbi fejlesztést. A megalkotásukhoz kiváló útmutató Robert C. martin, Clean Code : A Handbook of Agile Software Craftsmanship[27] című könyve!

2.4.8. Folyamatos integráció/Folyamatos szállítás (CI/CD)

"A folyamatos integráció olyan fejlesztési gyakorlat, amelyben a fejlesztők a kódot naponta többször egy közös felületen integrálják, egyeztetik. Így a kisebb változtatások is gyorsan elérhetővé válnak a csapat többi tagja számára. Az integrálást követően minden új kódrészlet ellenőrzésre kerül, amely lehetővé teszi a fejlesztők számára, hogy korán felismerjék a problémákat és még az elején javítsák azokat, így időt nyerve és minimalizálva a javítandó kódok mennyiségét.

A folyamatos szállítás a folyamatos integráció természetes kiterjesztése: olyan megközelítés, amelyben a csapatok biztosítják a rendszer gyors kiadhatóságát verzióként.

³A dolgozat későbbi részében részletes bemutatásra kerül.

A folyamatos szállítás célja, hogy minél előbb visszajelzést kapjunk a munkánkról, hogy azt minél tökéletesebben a megrendelő igényeihez igazíthassuk, és ezzel minél nagyobb elégedettséget érjünk el a megrendelőknél." [12]

A folyamatos integráció automatizált működése egy központi szerveren úgynevezett futtatók segítségével elindítja a szoftver menedzsment eszközt, ami felépíti a az applikációt, futtatja az általunk megírt teszteket valamint képes statikus kód analízist is végezni amennyiben azt előre definiáltuk. Ezek az analízisek segítenek rákényszeríteni a programozót a fejlesztés során a tiszta kód szabályainak betartására is.

Amennyiben folyamatos integrációt és folyamatos szállítást alkalmazunk képesek vagyunk olyan konfigurációra ami a megfelelő munkafolyamatok sikeres futtatása esetén a csővezeték utolsó munkafolyamataként automatikusan telepíti az applikációknak legújabb verzióját egy az általunk meghatározott szerverekre.⁴

2.5. A fejlesztéshez szükséges ismeretek összefoglalása

Ez a fejezet először bemutatta azokat a szabályozásokat amik segítségével részletes betekintést nyerhetünk a Magyar szabadságolás rendszerébe és megismerhettük annak összetettségét. Ezután általános képet kaphattunk az elérhető szolgáltatások és applikációkról működéséről, előnyeiről és hátrányairól, valamint a felhasználói igényekről, a Magyar Felsőoktatás szemszögéből. Ezen lehetősége költséges és kompromisszumokkal teli szolgáltatás nyújtásának ezekben a komplex szervezeti egységekben. Ezáltal elmondhatjuk, hogy egy saját célszoftver létrehozása nagyban segíthetné a szervezet részegységeinek munkáját, az igények és szabályok ismeretében képesek vagyunk ezt az applikációt megtervezni és megvalósítani. Végezetül felépítettük a tervezéshez és megvalósításhoz szükséges technológiai eszköztárat ami a rendelkezésünkre áll, és segítségével kivitelezhető egy webapplikáció ami nem csak kiszolgálhatja a korábban definiált elvárásokat, de akár igény szerint tovább is fejleszthető.

⁴A dolgozat későbbi részében részletes bemutatásra kerül.

3. fejezet

Automatizálható Webapplikáció Tervezése

A fejezet célja, hogy a korábban megismert fogalmak és elvárások segítségével, létrehozzunk egy alkalmazás csontvázát aminek az implementálásával egy jól strukturált könnyen karbantartható és fejleszthető megoldást valósíthassunk meg.

További fejlesztések esetén ezen dokumentáció alapján, akár új fejlesztők is betudjanak csatlakozni a megvalósításba.

3.1. Architektúra felépítése

3.1.1. Több rétegű alkalmazás

Az MNV ismertetése során felmerült már a Többrétegű alkalmazás architektúra, lényegében véve nem sokban különbözik egymástól a két rendszer struktúrája, csupán további rétegeket vezetünk be, annak érdekében, hogy alkalmazásunkban jobban elkülönítsük a rétegek szerepeit. Ebben a felépítésben viszonylag kötöttebb szabályok mentén kell haladnunk, de az adataink a rétegek közötti átalakítások miatt nagyobb biztonságban vannak.

Hátránya viszont, a kódsorok száma és a fejlesztési idő is jelentősen megnövekedhet az alkalmazásával.

A rendszer tervezett felépítése a 3.1. ábrán látható, ahol megfigyelhetjük, hogyan halad végig a különböző rétegeken a felhasználói kérés.

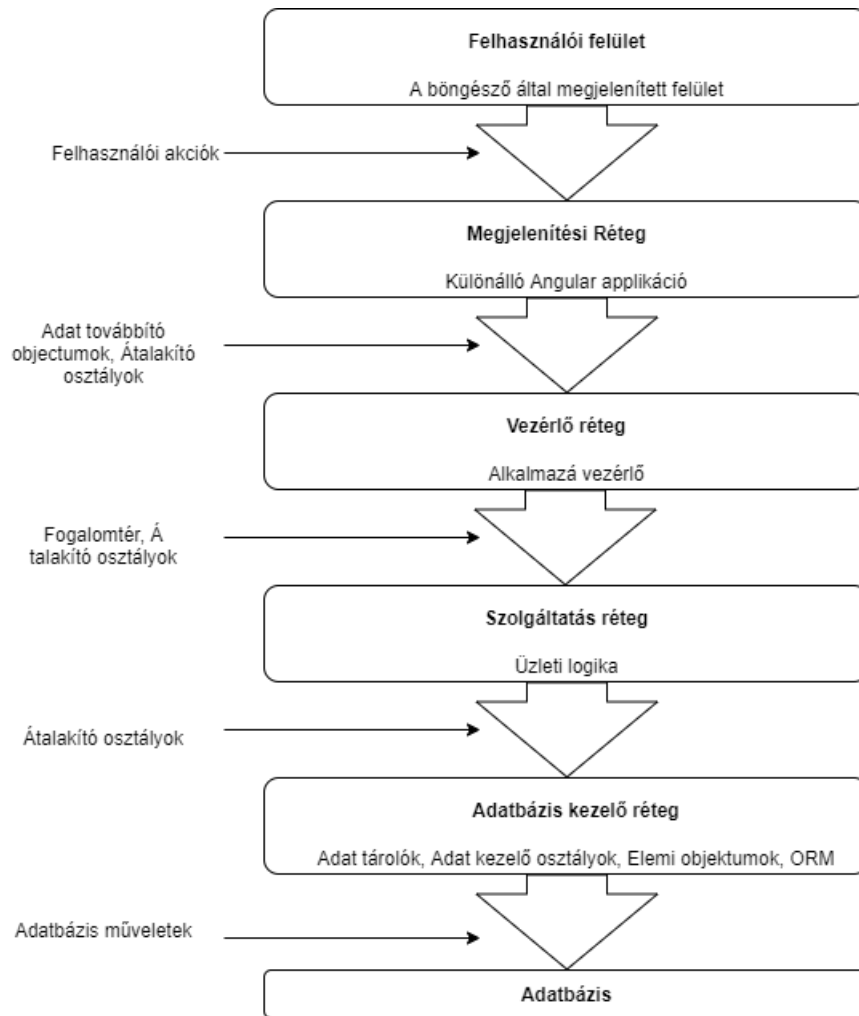
Az ábrán látható folyamat során a felhasználótól érkező adat feldolgozódik a megjelenítési rétegben, majd az általunk írt Angular applikáció egy JSON objektumot generál és egy HTTP híváson keresztül megszólítja a szerveren futó alkalmazást.

A háttérrendszer által definiált végponton a beérkező JSON kérést egy Adatközetítő objektummá alakítjuk¹, ezt az objektumot átalakítva átadjuk a szolgáltatási rétegnek.

Ebben a rétegben feldolgozzuk az adatokat, számításokat végzünk vele, majd a végső formájában átadjuk ismét egy átalakító osztálynak ahol Elemi objektummá alakítjuk.

Az adatkezelő rétegben, hozzáférünk az adatbázis kapcsolatainkhoz így az Elemi értékekből SQL-t létrehozva tudjuk meghívni az adatbázis egyes tábláit, ezzel beszúrni vagy akár lekérdezni adatokat azokból.

¹Ennek az objektumtípusnak a fontosságát később kifejtem.



3.1. ábra. Többrétegű alkalmazás architektúra.

3.1.2. Megjelenítési réteg

Az alkalmazásunk itt rendereli ki a felhasználó számára látható és elérhető weboldalt, ez a művelet jól elkülöníthető, akár külön kódbázis és infrastruktúra is tartozhat hozzá, amennyiben nem a már említett JSP technológiát vagy sablon alapú megjelenítést használjuk.

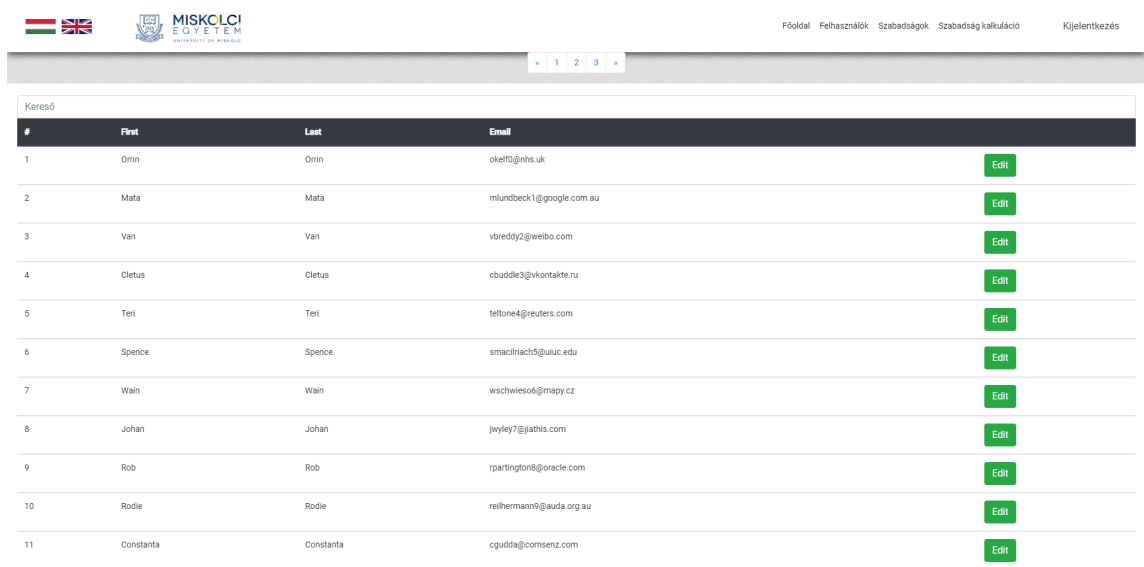
Az Angular használata mindenféleképp külön webszerver indítását igényli így érdemes ennek megfelelően különálló logikaként kezelni, akár egy önálló microalkalmazásként is tekinthetünk rá.

A programozási nyelv segítségével képesek vagyunk dinamikusan ellenőrizni a felhasználtól kapott bemeneti értékeket és nem megfelelő adat esetén rögtön jelezni az eltérést, ezzel segítve az elvárt működést.

Fontos, hogy a felhasználói felület letisztult és könnyen használható legyen, ugyanakkor az adatok kezelésében minden esetben a felület igazodik, a szerver oldali logikához. Nem itt definiáljuk az elküldött vagy elvárt JSON üzenetek tartalmát és felépítését. Több fejlesztő esetén a résztvevők közösen egyeznek meg az értékekről és a struktúráról.

Általános felület terv: A könnyen kezelhetőséget szem előtt tartva a 3.2. látható listanézetes megjelenítést érdemes használni az alkalmazás összes funkciójánál, hogy egyes adatokat megjelenítsünk.

Ennek a nézetnek a segítségével láthatóvá tesszük az elérhető adatokat, azokat az elhelyezett gombok segítségével szerkeszthetjük amik kattintás után az ehhez szükséges úrlaphoz navigálnak. Ezen az oldalon továbbá elérhetőnek kell lennie az elemi műveletekhez szükséges gomboknak mint például törlés, mentés.



#	First	Last	Email	
1	Orrin	Orrin	okeffo@nhs.uk	Edit
2	Mata	Mata	mlundbeck1@google.com.au	Edit
3	Van	Van	vbredy2@weibo.com	Edit
4	Cletus	Cletus	cbuddie3@vkontakte.ru	Edit
5	Teri	Teri	teftone4@reuters.com	Edit
6	Spence	Spence	smacilriach5@uiuc.edu	Edit
7	Wain	Wain	wschwieso6@magy.cz	Edit
8	Johan	Johan	jwyley7@jathis.com	Edit
9	Rob	Rob	rparrington8@oracle.com	Edit
10	Rodie	Rodie	reilhermann9@aoda.org.au	Edit
11	Constanta	Constanta	cgudua@comsenz.com	Edit

3.2. ábra. Általános felület terv

2

Az általános felület mintájára a következő típusú felületek megvalósítása szükséges:

1. Minden felhasználó számára elérhető.
 - Saját adatok megtekintése.
 - Felhasználható és felhasznált szabadnapok nyomon követése.
 - Szabadnapok kiírása és módosítása.
2. Jóváhagyó számára elérhető.
 - Vele relációban álló felhasználók szabadnapjainak nyomon követése.
 - Jóváhagyásra váró kérések listája, azokhoz tartozó műveletek.
3. Adminisztrátor számára elérhető
 - Szabadságolással kapcsolatos konfigurációs oldalak.
 - Vele relációban álló felhasználók létrehozása, adatainak módosítása.

Továbbá az adminisztrátor és a jóváhagyó számára egy statisztikai összesítéseket megjelenítő oldal, valamint egy éves szabadságolási terv létrehozására és módosítására alkalmas naptár nézettel rendelkező oldal megvalósítására is szükség van.

²A jogkörökről és azokhoz tartozó funkciókról a dolgozat későbbi részében lesz szó.

3.1.3. Vezérlő réteg, és Adatközzvetítő objektumok

Az alkalmazásunk szerver oldali megvalósítása a vezérlőrétegnél kezdődik, itt definiálhatjuk a végpontokat amik segítségével kommunikálhatunk a közvetlen felettünk lévő réteggel. Ezek a végpontok URL paraméterként jelennek meg, így könnyedén hivatkozhatunk rájuk.

Vezérlőfüggvények Minden függvényhez rendelünk egy URL értéket amin keresztül a függvény meghívódik, az argumentumaink attól függően változhatnak milyen HTTP hívásnak feleltetjük meg a függvényt.

Elemi adattípusok esetén az URL paramétereként adjuk át a kívánt információt, objektumot viszont JSON kérésként kezelünk. Ahhoz, hogy a beérkező JSON értékből számunkra kezelhető objektum legyen szükségünk van egy Adatközzvetítő objektumra.

Ezek a függvények minden esetben átadják az adatot egy átalakító osztálynak ami egy Belső objektumot hoz létre és tovább adja azt a megfelelő szolgáltatás osztálynak.

Adatközzvetítő objektumok Kifejezetten fontos, hogy a beérkező adatokat könnyen tudjuk ellenőrizni, ezeknek az osztályoknak a használatával előre meghatározzuk a számunkra szükséges értékeket, és könnyedén leellenőrizhetjük a beérkező adatokat.

3.1.4. Szolgáltatás réteg, és Belső objektumok

A működéshez szükséges logika megvalósítását végezzük itt, a beérkező adatok feldolgozása mellett a számolások és az automatizált eseménykezelés is itt kerül implementálásra.

Az adatbázis védelme érdekében a műveleteket belső objektumokon hajtjuk végre, amint elnyerték végleges állapotukat hívási iránytól függően vagy átadjuk a vezérlésnek megjelenítésre, vagy átalakítjuk Elemi objektummá, majd továbbítjuk az adatkezelési rétegnek.

3.1.5. Adatbázis kezelő réteg, ORM, és Elemi objektumok

Adatkezelő osztályok Ezek lesznek a kapcsolódási pontjaink az üzleti logika és az adatbázis között. Fontos, hogy ezek az osztályok csak Elemi objektumokat kezeljenek, csak és kizárólag a szolgáltatás rétegnek adhatnak át Belső objektumot.

Elemi objektumok Az adatbázis tábláinak reprezentációja, az osztályban definiált mezők és a tábla oszlopainak elnevezése mindenképp meg kell egyezzen! Ezen felül ebben az osztályban is definiálnunk kell a relációs kapcsolatokat nem csak a táblákat létrehozó SQL parancsokban!

3.1.6. A hívási lánc összegzése

Összefoglalva a hívási lánc folyamán minden esetben legalább két átalakítást végzünk, egymással akár identikus objektumokon esetén is, ez nehezíti a fejlesztést, és megnöveli a kérések kiszolgálásának az idejét, de ez egy elhanyagolható probléma azzal a biztonsággal szemben amit az architektúra nyújt.

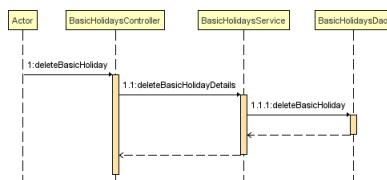


Lekérdező hívás esetén kihagyható a Belső objektumból Adatközvetítő objektumra való átalakítás, abban az esetben ha a belső objektum nem tartalmaz felesleges információt. Lényeges, hogy csak a szükséges adatokat továbbítsuk a megjelenítési réteg felé, felesleges adatok kiküldése szintén biztonsági problémát okozhat és sokkal jobban lassíthatja a megjelenítést mint a szerver oldali átalakítások.

Az ehhez tartozó szekvenciadiagram látható a 3.4. diagramon.



Törlés esetén csak az adott elem azonosítóját várjuk paraméterként amit a gombnyomás automatikusan be tud küldeni, így validációra nincs szükség ezáltal a szekvencia is leegyszerűsödik. A törlés szekvenciája a 3.5. diagramon látható.



3.5. ábra. HTTP Delete kérés szekvenciája

3.2. Adatbázis struktúra

Webalkalmazások esetén az idő és erőforrás költség nagy részét az adatbázis műveleteken teszik ki, ezért fontos, hogy egyszerű és jól karbantartható struktúrát alakítsunk ki.

Az alkalmazáshoz tervezett adatbázis logikailag három részre bontható, kisebb önálló konfigurációs táblák, felhasználókhöz tartozó adatok tárolása, valamint a jóváhagyási funkcióhoz szükséges táblák csoportja.

Minden logikai egység külön tervezési szemléletet igényel, ennek megfelelően az egy-ségenkénti reláció megengedett, de az egységek közötti kapcsolatok kiépítése könnyen hibához vezethet.

3.2.1. Konfigurációs táblák

A korábban ismertetett szabadságolási szabályoknak megfelelően definiáltunk alap és pótszabadságot, valamint a pótszabadságon belül felsoroltunk több lehetőséget is, ezen lehetőségeknek saját értékeik és feltételeik vannak.

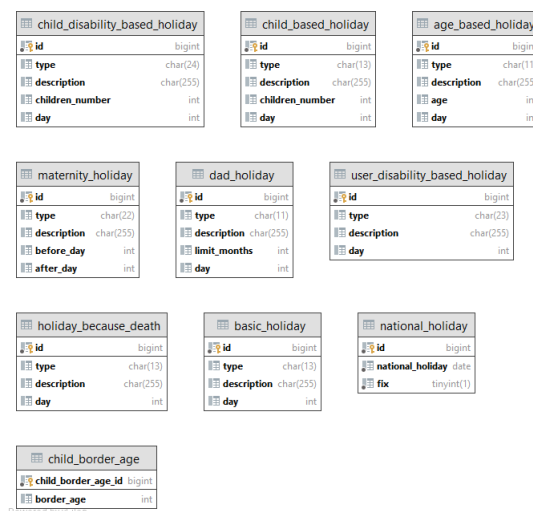
Alapvetően minden típushoz tartoznia kell egy egyedi azonosítónak és egy szám értéknek ami a szabadnapokat reprezentálja, ebből a tényből kiindulva érdemes lenne egy táblában tárolni az összes típust és megkülönböztetni azokat az egyedi értékeik alapján.

Ezzel a megoldással a következő problémák merülnek fel:

- Az egyedi szabályozásokhoz saját értékek tartozhatnak, ebben az esetben komplex több oszlopos táblát kell létrehoznunk amiben kifejezetten sok null értéket tárolunk el.
- Minden lekérdezéskor összetett keresést kell alkalmaznunk, ha adott szabadság-típust keresünk.
- A tábla karbantartása komplikált feladat, ami redundanciához, és teljesítmény-romláshoz vezethet.

Ezáltal a tervezés során arra döntésre jutottam, hogy minden szabadság típus önálló adatbázistáblában definiálok, ahogy az a 3.6. diagramon látható. Ezáltal sok kisméretű táblát hozok létre, ezzel biztosítok jobb teljesítményt és költséghatékonyabb működést az applikációmnak ezen a területén.

Továbbá a táblák tartalmát képezi egy típus érték is ami minden esetben konstansként jelenik meg és a pótszabadság típusát jelzi, ezzel az értékkel az automatizálási folyamat alatt megtudjuk különböztetni egymástól a típusokat.



3.6. ábra. Konfigurációs táblák UML diagramja

Valamint minden tábla kapott egy leírás mezőt melyben a konfiguráció során elnevezhetjük az adott sort, hogy az egyes sorokat a felhasználó a felületen keresztül könnyebben megkülönböztesse.

Az időarányos szabadnapok számolásához, szükség van még a nemzeti ünnepek tárolása is, a tábla értéke megjelölhető egy boolean segítségével.³

Az alap szabadság esetén az adminisztrátor konfigurálhat egy 20 munkanapos szabadságot az "átlag" leírással, valamint egy 21 munkanapos sort is felvehet a "közalkalmazott" értékkel.

3.2.2. Felhasználói adatok

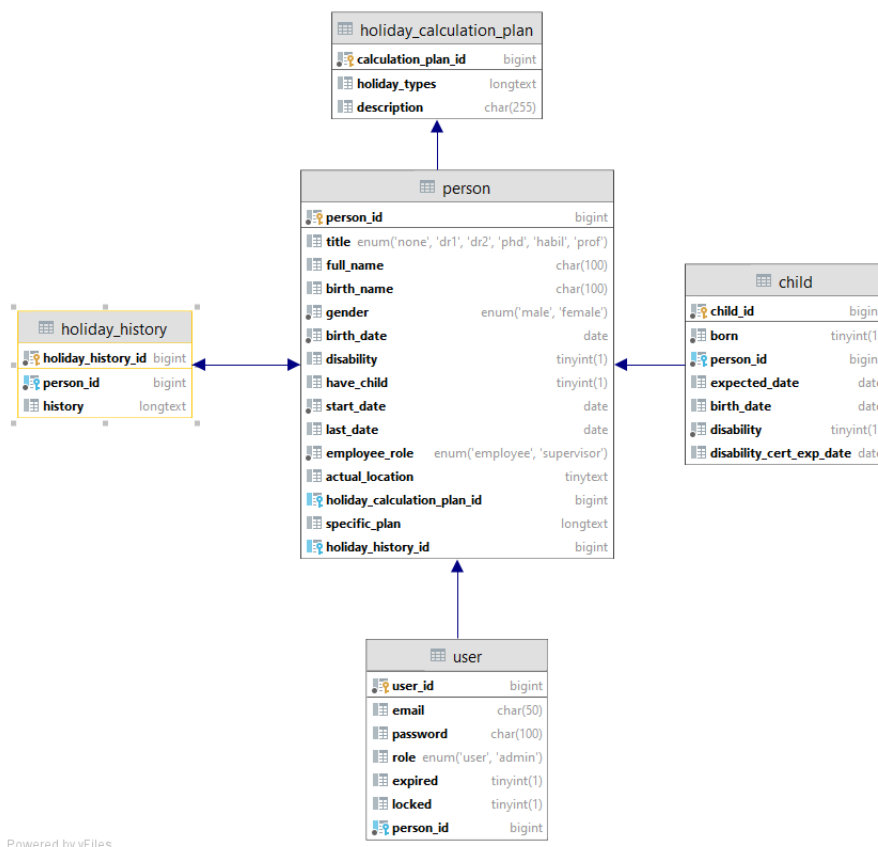
Az egység célja, hogy csak és kizárólag olyan adatot tároljunk a felhasználóról ami feltétlenül szükséges lesz az automatizáláshoz, mivel léteznek olyan pótszabadságok amik ha a jogszabály nem is változik maga a szabadság értéke függ a felhasználó bizonyos változó adataitól, ezért automatizálás szempontjából a változó adatokat állandó értékekből érdemes számolni a programban.

A logikai egységet a 3.7 diagramon láthatjuk, a táblák egyéni szerepe:

- **user:** A felhasználó beléptetése során az azonosítást szolgálja
- **person:** A felhasználó személyes adatai amik kalkulációkhoz és automatizáláshoz szükségesek.
- **child:** Ez a tábla egy több kapcsolatban áll a person táblával, mivel egy felhasználónak több gyermeke is lehet ennek függvényében eltérő pótszabadságokra jogosult
- **holiday_calculation_plan:** Ebben a táblában az adminisztrátor által meghatározott számítási formák vannak tárolva.⁴, ez a tábla is egy több kapcsolatban

³A boolean jelölés szerepét az implementáció alatt részletezem

⁴A fejezet későbbi részében részletesen kitérek rá.



3.7. ábra. Felhasználóval kapcsolatos táblák UML diagramja

áll a person-nel itt viszont, egy holiday_calculation_plan-hez tartozhat több felhasználó.

- **holiday_history:** Mindenki egyéni szabadságolási története⁵, éves felbontásban tárolja az egyénekre vonatkozó szabadságolási adatokat.

Érdemes még kitérni a "person" tábla néhány elemére:

- **disability:** Pótszabadság illeti az alkalmazottat amennyiben valamilyen tartós egészségkárosodással él, mint például erőteljesen látás sérült.
- **have_child:** A szülői státusz gyors megállapítását hivatott biztosítani.
- **start_date:** A munkaviszon kezdeti időpontja.
- **last_date:** A munkaszerződés utolsó napja. (Opcionális érték)
- **employee_role:** A felettes beosztott viszony definiálásában segít.
- **specific_plan:** Az általános holiday_calculation_plan speciálisan a felhasználóra alakított változata.⁶

Mindezek mellett, a "child" tábla is összetett adatokat tartalmaz:

⁵A fejezet későbbi részében részletesen ismertetem

⁶A dolgozat későbbi részében részletesen bemutatom.

- **born:** Megmutatja, hogy a szóban forgó gyermek, megszületett-e már.
- **expected_date:** A születés várható időpontja (Csak akkor létezik, ha `born = false`).
- **birth_date:** A születési dátum, több felhasználási módja is van (Csak akkor létezik ha `born = true`).
- **disability:** Amennyiben a gyermek tartós fogyatékkal él a szülőt pótszabadság illeti.
- **diability_cert_exp_date:** A szülő köteles bemutatni a gyermeke tartós hátrányos helyzetéről szóló határozatot, ennek egyes esetekben van lejárat ideje.

3.2.3. Összetett adatok tárolása kisméretű táblákban

Bizonyos objektumok túlságosan összetettek ahhoz, hogy ezeket egy a már bemutatott egyszerűsített adatbázisban tároljunk, amennyiben összetett egy több kapcsolattal tele-tűzdelt adatbázisrendszerben valósítjuk meg, az könnyedé vezethet rendkívül bonyolult struktúrát, és hosszan tartó komplex lekérdezéseket.

Megfigyelhető a 3.7 diagramon, hogy a **holiday_types**, **specific_plan**, és a **history** mezők longtext érték definíciót kaptak. Ez a tervezési döntés azért született, hogy az applikáció Belső objektumaiban bonyolult osztályhierarchiában álló adatokat egyetlen JSON-ben tudjam tárolni.

Ezzel a megoldással több tábla és több-száz sornyi adatbázis rekord létrehozását és tárolását kerülöm el, nem csak a nagy mennyiségű tárhelyet csökkentem hanem a lekérdezések bonyolultságát is, ezzel együtt az applikáció teljesítménye is növekszik.

Fontos megemlíteni, hogy ez a tervezési döntés nagyban hozzájárul az automatizálás megvalósításához, mivel az időigényes adatbázis műveletek lecsökkennek egyetlen lekérdezésre, aminek a végén a String típusú eredményből könnyen kezelhető objektum keletkezik.

3.2.4. Jóváhagyáshoz szükséges táblák

Ma a szoftverfejlesztésben gyakori igényt képez, hogy a felettesek jóváhagyják beosztottaik különböző kéréseit, ez önmagában komoly tervezést igénylő probléma, viszont megoldható egy létező ingyenes API hívással ami a saját adatbázisunkban dolgozik. A kiszolgáló legenerálja a számára szükséges táblákat és menedzseli azokat.[1]

3.3. Funkció tervezés

Az elvégzett igényfelmérés, és a már létező megvalósítások alapján részletesen specifikálhatóak azok a funkciók amelyek teljes megvalósítása esetén az applikáció teljesen elkészültnek minősíthető. Lényeges megemlíteni, hogy amennyiben a megvalósítás során megfelelő fontossági sorrendet tudunk állítani elérhető az a pont a fejlesztésben, ahonnan a program üzembe helyezhető. Ezzel már nagyban könnyíteni a felhasználók munkáját, amíg a további fejlesztések folynak.

Minden itt leírt funkció esetén megvalósításnál a fejezet elején lefektetett architektúra követése az elvárt működés része.

3.3.1. Felhasználó és hozzáférés kezelés

A szoftverbiztonság kiemelten fontos még a belső hálózaton működő rendszereknél is, viszont amennyiben a Webapplikációnk nyilvános címen üzemel a hozzáférés kezelés elengedhetetlen.

A program ezzel kapcsolatos működési követelményei:

1. A felhasználó egyedi felhasználónév és jelszó segítségével be tud jelentkezni a felületre, továbbá van kijelentkezési lehetősége is.
2. Minden felhasználó csak és kizárólag a jogkörének megfelelő oldalakhoz és funkciókhoz fér hozzá.

Adminisztrátor jogkör Többnyire az intézeti/tanszéki adminisztrátor tölti be ezt a szerepet, ehhez a jogkörhöz tartozik, a legtöbb hozzáférés és funkció.

Felhasználó létrehozása Alapvető adminisztrátori feladat, amely során a külön erre létrehozott menüpont alatt az ügyintéző megadja a következő felhasználói adatokat:

- egyedi email címe,
- teljes neve,
- jogköre.

A létrehozás gombra kattintva, a program létrehozza az adatbázisban az új felhasználót.

Felhasználó adatainak lekérdezése és módosítása Az adminisztrátor megnyithatja az a menüpontot ahol kilistázzuk az összes alkalmazottat, a listában tud keresni, megfelelő gombra kattintva tovább navigáljuk a felhasználó részletese adatait tartalmazó oldalra, ahol a következő funkciókat tudja használni:

- felhasználói adatok szerkesztése és mentése,
- felhasználó gyermekeire vonatkozó adatok szerkesztése és mentése,
- adatok törlése,
- felhasználóhoz kapcsolódó szabadságra vonatkozó adatok hozzárendelése, generálása (specializált szabadság szabályzat).

Felhasználóra vonatkozó szabadságolási szabályok létrehozása Az adminisztrátor rendelkezik olyan funkcióval ahol menedzselhet szabadságolási szabályokat, amely tartalmazza a kiválasztott alap szabadságot, és a felhasználókra vonatkozó pótszabadságok típusainak listáját.

Felhasználó, felettes jogkör Ebben a jogkörbe tartozik a szervezeti egység vezetője, lekérdezheti az beosztottak listáját, és azok szabadság történetét.

Alkalmazotti kérések bírálása A felettes rendelkezik olyan menüponttal, ahol lista nézetben láthatja a beosztottak szabadságolással kapcsolatos kéréseit, azokon a következő műveleteket tudja végezni:

- Kérelem elfogadása,
- Kérelem visszautasítása,
- Visszautasítás indoklása.

Éves szabadságterv készítése és módosítása Elérhető egy felület ahol naptár nézetben jól elkülöníthető színekkel jelezve könnyen szerkeszthető és menthető az éves szabadságolási terv.

Felhasználó, beosztott jogkör A legtöbb felhasználó ebbe a jogkörbe tartozik, az általuk látható és használható funkciók a legkevesebb.

Saját adatok lekérdezése A felhasználó bármikor lekérdezheti, de nem módosíthatja az adatokat amiket az adatbázisban tárolunk róla.

Saját szabadnapok kiírása, nyomon követése A korábban definiált szabályoknak megfelelően képes létrehozni szabadnapokat, amelyek a felettesi bíráláson keresztül mennek, a már létrehozott szabadnapok módosítása újbóli bírálásra kerül.

A felületen megtekintheti az éves szabadnapjainak a számát valamint a még fel nem használt napok is kijelzésre kerülnek.

Fontos, hogy ezek a funkciók a többi jogkörben is elérhetőek!

3.3.2. Szabadságolási szabályok konfigurálása

Kulcsfontosságú funkció ami az adminisztrátori jogkör alá tartozik.

Alap és pótszabadságok Minden szabadságtípushoz tartozik egy egyedi, különálló felület ahol a szabályoknak megfelelően szerkeszthetőek, minden esetleges további konfiguráció jól érthető külön felületen van létrehozva.

Alapszabadság konfigurálásához a felhasználó a következő lépéseket kell végrehajtassa:

1. Elnavigál az alapszabadságok menüpont alatt a listanézetre
2. A hozzáadás gombra kattintva átirányítjuk az alapszabadság létrehozása űrlaphoz.
3. A felület kitöltése után rákattint a mentés gombra.
4. Sikeres végrehajtás esetén átirányítjuk a listanézetre, ellenkező esetben a megfelelő hibaüzenettel tájékoztatjuk.

Nemzeti ünnepek Ezen dátumok ismerete fontos részét képezik az éves szabadság arányos számolásának, rendkívül fontos az automatizálás szempontjából, hogy minden évben a valóságot tükrözze az általunk tárolt adat.

Az elvárt működésnek több megvalósítási lehetősége van:⁷

1. Az adminisztrátor kézzel konfigurálja ezeket az adatokat minden évben.
2. A manuális konfiguráció során megjelöljük a nem mozgó dátumokat amit automatikusan újra tudunk számolni, minden más évente kézzel konfigurál a felhasználó.
3. Az állandó dátumokat SQL segítségével beégetjük az adatbázisba, a mozgó dátumokat újraszámoljuk.
4. Külső API hívás segítségével lekérdezzük minden év elején az adatokat amikre kíváncsiak vagyunk.

3.3.3. Automatizált működés

A rendelkezésre álló információk alapján az automatizálható funkciók főként az éves kalkulációkkal és tervezési munkálatokkal kapcsolatosak, ezeken felül viszont mindennapos automatizált futással a gyermekszületéssel kapcsolatos adatok alapján automatizálható az ehhez kapcsolódó esemény alapú pótszabadságok kezelése is.

Éves szabadnapok időarányos számolása Ez a funkció csak részben automatizálható, amennyiben az adminisztrátor év közben hoz létre felhasználót abban az esetben biztosítanunk kell egy gombot számára amivel lehetősége nyílik az azonnali számolások elindítására.

Az automatizálással kapcsolatos műveletek a következők:

1. Minden év utolsó napjának a végén a megfelelő számolást és módosítást végző funkciók futása,
2. A felhasználóhoz rendelt szabályok, specializált formájának aktualizálása.
3. Az aktualizált szabályok alapján az éves szabadság kiszámolása.
4. Az éves szabadság arányos újra számolása tört év esetén.
5. Az új éves történet létrehozása, és az éves szabadságkeret hozzáadása.
6. Szabadság történet frissítése és mentése a megfelelő felhasználóhoz csatolva.

Születéssel kapcsolatos szabadnapok esemény alapú kezelése Ez az automatizálás napi szintű futtatást igényel ezeket az ellenőrzéseket nap végén érdemes elvégezni.

A tervezett működés lépései a következők⁸:

1. Az érintett alkalmazottak kikeresése.
2. A szabályozásnak megfelelő szabadságok létrehozása

⁷A megvalósítás alatt ezekkel a lehetőségekkel részletesen foglalkozok.

⁸Férfiak és nők esetén a szabályozások különböznek így a megvalósításban pontosabb képet kaphatunk a témáról.

3. Az éves történet módosítása, és mentése.
4. Esetleges utólagos ellenőrzések elvégzése.

Éves szabadságterv generálása Ennek a funkciónak az automatizálása, a legösszettebb, mivel nem csak az országos szabályozásoknak de az intézmény szabályainak is eleget kell tennünk.

Az automatizált futásnak minden év december 31. előtt meg kell történnie, a tervezett lépései:

1. Az összes felhasználó lekérdezése.
2. A felhasználók adatainak frissítése.
3. Éves szabadság egyenlegük arányos kiszámolása.
4. A szabadnapjaik beosztása a 7-14 napos szabálynak megfelelően.
5. Az egyetemi szabályzatban előírtak betartása.
6. Szabadságterv mentése egy saját adatbázis táblába.

3.3.4. Szabadságolási statisztikák készítése

Az adminisztrátor jogkörrel rendelkező felhasználónak elérhető, egy menüpont az oldalon, ahol statisztikákat tud generálni az intézet alkalmazottainak adatai alapján.

A funkcióban a következő működések szerepelhetnek:

- Statisztika kimenetének kiválasztása (diagramok, táblázatok, esetleg formanyomtatványok),
- Adatok intervallumának dinamikus állíthatósága,
- A statisztikák alanyainak beállítása.

4. fejezet

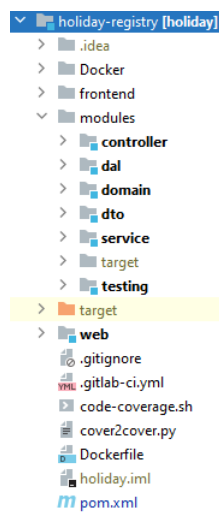
Az alkalmazás megvalósítása Java Enterprise környezetben

4.1. Mappastruktúra kialakítása

A jól strukturáltság fontos szempont az implementáció alatt, mivel az osztályok konzisztens elnevezésével funkciónak megfelelő elhelyezésével a fejlesztési folyamat során könnyebben azonosíthatjuk az elhelyezkedésüket.

4.1.1. Modularizált alkalmazás

Az alkalmazásunk viselkedés alapján való modulokra bontása jól elkülöníthetővé teszi a különböző szerepkörű implementációkat, és a maven segítségével biztosíthatjuk, hogy a modulok lássák egymást.



4.1. ábra. Modularizált alkalmazás mappastruktúrája

A már részletesen ismertetett Többszintű alkalmazás felépítését tükrözi a 4.1. ábra. Az itt látható felosztásban a gyökér könyvtárban belül található a web module aminek jelen esetünkben csak konfigurációs szerepe van, mivel a spring-boot alkalmazások esetén a main függvényt tartalmazó osztálynak a csomaghierarchia legkülső csomagjában kell elhelyezkednie.

Amennyiben ez nem megfelelő helyen található az alkalmazásunk nem fogja látni a létrehozni kívánt komponenseket és a kontextust nem tudja felépíteni.

Frontend mappa Szintén leolvasható a 4.1. ábráról, hogy a jelen esetben struktúra részét képezi a megjelenítési réteghez szükséges applikációt tartalmazó mappa is. Mivel jelenleg nem microservice architektúrát alkalmazunk ez a megoldás is elfogadható, de gyakorlatban nem ajánlott és igen ritka.

Testing Ez egy úgynevezett rétegeken átívelő modul, aminek minden rétegben szerepe van, ebben a megvalósításban a modul nem tartalmaz kódot, csupán konfigurációt és a maven jacoco nevezetű teszt riport generáló kiegészítője által létrehozott fájlokat. Ezeket a fejlesztés során a tesztlefedettség méréséhez használjuk, képes vizualizációi segítségével megmutatni egy html oldalon, melyik osztályban milyen szintű a lefedettség, és azt is milyen hiányosságaink vannak tesztek terén.

4.2. Összetett objektum előállítása JSON formátumból Lombok használatával

A szoftverfejlesztés egyik alapelve az, hogy ne ismételd önmagad, ami a redundanciából eredő hibalehetőségek és fejlesztési komplikációk csökkentése érdekében lett egy kitűzött cél. Az elv ellenére mégis rengeteg egyszerű kódsort ismételünk újra az Objektum Orientált fejlesztés során.

4.2.1. Az ismétlődő programrészletek

Minden általános java objektum létrehozásakor létrehozunk az osztály mezőiihez:

- legalább egy konstruktort,
- beállító és lekérdező függvényeket,
- titkosító valamint egyenlőség vizsgáló függvényeket,
- és a könnyebb hibakeresés érdekében, még egy kiíró függvényt is.

Ezek az implementációk legtöbbször annyira elemiek, hogy minden fejlesztőkörnyezet, automatikusan képes generálni a hozzájuk tartozó implementációt. Ez a sokszor több száz sornyi kódrészlet rengeteg tárhelyet jelent valamint tesztelés szempontjából ezek az osztályok többnyire rontják a lefedettséget. Mivel ezeknek a függvényeknek a tesztelése a nyelv működésére irányulna, így a saját alkalmazásunkban felesleges ezekre tesztet implementálni.

4.2.2. A Lombok működése

A felvázolt probléma megoldására született a Project Lombok[2] könyvtár aminek a segítségével egyszerű annotációk használatával használhatjuk ezeket a kódrészleteket anélkül, hogy implementálnánk azokat a saját programunkban

@Data használatával a mezők definiálása után rögtön használhatóvá válnak a már említett redundánsnak mondható kódok.

```
@Data
public class UserEntity {
    private Long userId;
    private String email;
    private String password;
    private UserEntityRole role;
}
```

4.2.3. Megváltoztathatatlan objektumok Lombokban

Objektum orientált programozásban gyakran használunk objektumokat adatok továbbítására, adattovábbítás esetén fontos, hogy gondosodjunk az objektum állapotáról. Amennyiben az változtatható a programozás, abban az esetben megnövekszik a hibalehetőségek száma.

Ahhoz, hogy ne lehessen bármikor megváltoztatni egy objektum mezőjének az állapotát, úgy kell megírni a hozzá tartozó függvényeket, hogy a mezőértékek csak és kizárólag a referencia létrehozásakor legyenek beállíthatóak.

A megoldás egy része igen egyszerű nem, hozunk létre olyan függvényeket ami példányosítást követően módosítani tudja az állapotot, tehát a beállító vagy más néven set függvényeket nem implementáljuk.

Ezáltal, viszont konstruktor argumentumokon keresztül lehet csak példányosítani, és ha esetleg adott esetben az egyik paraméter nem létezik ott, a példányosításkor null értéket kell átadnunk. Továbbá a paraméter sorrend is gondot okozhat ezért ez egy körülményes megoldás.

Építő minta segítségével pontosan láthatjuk melyik mezőt definiáljuk és melyiket nem, ebben a megoldásban egy privát konstruktoron keresztül példányosítjuk az osztályunkat aminek a belső builder azaz építő osztályán keresztül állítunk be.

Ez ismét a korábban említett problémakörhöz vezet, ezért a Lombok könyvtár erre is nyújt megoldást a **@Value** valamint a **@Builder** előlések használatával.

Amíg az előbbi a megváltoztathatatlan osztály feltételeit teremti meg addig az utóbbi segítségével egy könnyen használható építő mintát megvalósító belső osztályt kapunk.

4.2.4. JSON felépítésű String generálása Jacksonnal

A Jackson könyvtár egy olyan megvalósítást kínál ami segítségével könnyedén alakíthatjuk át összetett java objektumainkat egy egyszerű JSON formátumú String értéké.

Ennek a könyvtárnak a segítségével Belső modulban létező Holidayhistory.java osztályban található összetett objektum könnyedén átalakítható egy egyszerű String típusú és bármikor elvégezhető az átalakítás visszafelé is.

A következő osztály struktúrából:


```
public class HolidayHistory {
    Long id;
    PersonDetails person;
    List<YearHistory> yearHistories;
}

public class YearHistory {
    int year;
    BigDecimal fixHolidaysForTheYear;
    List<EventRelatedHoliday> eventRelatedHolidays;
    List<HolidayEvent> holidayEvents;
}

public class EventRelatedHoliday {
    EventRelatedHolidayType holidayType;
    int days;
    LocalDate eventDate;
    LocalDate expiration;
}

public class HolidayEvent {
    String description;
    LocalDate startDate;
    LocalDate endDate;
    Boolean dadHoliday;
}
```

Az itt látható JSON értéket generálja ami már könnyedén menthető az adatbázisunkban.

```
[
{
    "year": 2020,
    "fixHolidaysForTheYear": 10,
    "eventRelatedHolidays": [
        {
            "holidayType": "dad",
            "days": 0,
            "eventDate": "2021-03-30",
            "expiration": null
        },
        {
            "holidayType": "dad",
            "days": 0,
            "eventDate": "2021-03-30",
            "expiration": "2021-04-10"
        }
    ]
},
]
```

```

        "holidayEvent": [
            {
                "description": "desc",
                "startDate": "2020-03-04",
                "endDate": "2020-05-20",
                "dadHoliday": null
            }
        ],
    },
    {
        "year": 2021,
        "fixHolidaysForTheYear": 10,
        "eventRelatedHolidays": [
            {
                "holidayType": "dad",
                "days": 0,
                "eventDate": "2021-03-30",
                "expiration": null
            },
            {
                "holidayType": "dad",
                "days": 0,
                "eventDate": "2021-03-30",
                "expiration": "2021-04-10"
            }
        ],
        "holidayEvent": null
    }
]

```

A Jackson elap beállítás esetén a JSON létrehozásához a lekérdező függvényeket használja, így egyértelmű, hogy a JSON formátumból a beállító függvények segítségével állít elő objektumot.

Ahhoz, hogy aombok és Jackson könyvtárak megfelelően működjenek együtt Megváltoztathatatlan objektumok létrehozása esetén az építő függvény további beállításaira valamint új annotációk használatára van szükségünk.

Minden érintett osztályt a következő alakra kell hozni.

```

@Value
@Builder(builderClassName = "YearHistoryBuilder",
access = AccessLevel.PUBLIC)
@JsonDeserialize(builder =
YearHistory.YearHistoryBuilder.class)
@AllArgsConstructor(access = AccessLevel.PRIVATE)
public class YearHistory {

    @JsonProperty("year")
    int year;
    @JsonProperty("fixHolidaysForTheYear")

```

```

BigDecimal fixHolidaysForTheYear ;
@JsonProperty ("eventRelatedHolidays")
List<EventRelatedHoliday> eventRelatedHolidays ;
@JsonProperty ("holidayEvent")
List<HolidayEvent> holidayEvents ;

@JsonPOJOBuilder(withPrefix = "")
public static class YearHistoryBuilder {}
}

```

Az itt látható alakban a Jackson könyvtár már képes azonosítani az átalakításhoz szükségesombok által biztosított függvényeket.

4.3. Automatizált Cron munkamenetek Springben

Az automatizálás kulcsfontosságú szerepet játszik az alkalmazás piaci rést betöltő lehetőségeiben, ezért kifejezetten fontos, hogy a beállított időzített processzek fond nélkül végrehajthódnak.

Java timer segítségével képesek vagyunk másodpercben definiálni egy időzítőt ami az első indítástól számítva minden alkalommal folyamatosan számol és az időzítő lejáratkor elindíthatunk egy feladatot vagy komplex feladat sort is.

Sajnos ez a megoldás nem megfelelő számunkra hiszen az időzítő minden újrainduláskor lefutna valamint akár kiszámíthatatlan viselkedést is generálhat.

4.3.1. A keretrendszer nyújtotta előnyök

A Spring keretrendszerben sok megbízható és könnyen alkalmazható megoldás található már létező problémákra, ezek közé tartozik az automatizált függvényhívások is.

A megoldás során a keretrendszer egy Cron munkamenetet, hoz létre az általunk konfigurált futási feltételeknek megfelelően, ez esetben nem az applikáció belső számlálója felel a függvényhívás elindításáért. A munkamenet a szerver számítógépen jön létre és minden alkalommal meghívja azt a függvényt amelyiken definiáltuk amennyiben az aktuális dátum és idő egyezik az általunk konfigurálttal.

4.3.2. Cron munkamenet implementálása Spring-boot-ban

A spring-boot applikáció indító függősége automatikusan tartalmazza a számunkra szükséges könyvtárakat, viszont az eléréséhez és használatához meg kell jelölnünk a megfelelő osztályokat és függvényeket.

```

@SpringBootApplication
@EnableScheduling
public class HolidayApplication {...}

```

Engedélyezést követően az applikáció kontextusában bármelyik nyilvános függvényünk megjelölésével létrehozhatunk egy új munkamenetet.

```
@Scheduled(cron = "1 0 0 * * ?")
public void schedule() {
    unusedHolidaysService.addPenalty();
    childBornHolidaysService.validate();
}
```

Léteznek előre definiált beállítások a automatizált futtatáshoz, de ahogy a kódrészlet is mutatja bármikor konfigurálhatunk saját feltételeket, a cron leírónyelv segítségével.

A nyelvben hat karaktert kell megadnunk amik jelentését a 4.1 táblázat tartalmazza:

4.1. táblázat. Cron leírónyelv értékei.

Érték	Kötelező	Megengedett értékek	Speciális karakterek
Másodperc	Igen	0-59	, - * /
Perc	Igen	0-59	, - * /
Óra	Igen	0-23	, - * /
Hónap napja	Igen	1-31	, - * ? / L W C
Hónap	Igen	0-11 vagy JAN-DEC	, - * /
Hét napja	Igen	1-7 vagy SUN-SAT	, - * ? / L C #
Év	Nem	üres vagy 1970-2099	, - * /

4.4. Automatizált szabadság számolás tört év esetén, Nemzeti ünnepek tárolása

Az automatizálás folyamatát és a törvényben leírt szabályoknak megfelelő számítást ismerve könnyedén megvalósíthatunk egy összetett hívási láncot amely végén az éves vagy akár a részarányos szabadnapok számolása megtörténik. A kihívást jelentő probléma az egyenletben egyedül, az aktuális évben hátralévő munkanapok száma jelenti, ehhez ugyanis minden számolás esetén tudnunk kell a tárgy évben mennyi olyan munkanap létezik amely az aktuális nemzet számára munkaszüneti napnak minősül.

Habár a szóban forgó dátumok többsége állandó és az ezekre vonatkozó megoldás kézenfekvő, mozgó ünnepek esetén nem tárolhatunk állandó értéket.

4.4.1. Külső API hívása

Kiseb kereséssel rengeteg számunkra hasznos szolgáltatást találhatunk ahol, egy bizonyos URL összeállításával és a mögötte található végpont meghívásával minden szükséges információhoz hozzájuthatunk.

A legegyszerűbb verzió visszatérési értéke teljes mértékben tökéletes lenne számunkra hiszen tartalmazza azt a számot amit felhasználva csak be kell helyettesítenünk az egyenletünkbe.

Több lehetőség van amelynél az adott év vagy nap adatainak megadása esetén visszakapjuk, hogy az munkanap vagy sem.

A legnagyobb probléma viszont ezekkel az elérhető alkalmazásokkal, hogy az ingyenes hozzáférésük szigorúan korlátozva van, havi vagy éves előfizetésük pedig jelen esetben nem összeegyeztethető ár érték arányban az igényeinkkel.

4.4.2. Mozgó ünnepek számolása

A külső API hívások nélkül a legkézenfekvőbb megoldása az automatizálásnak, hogy saját magunk számoljuk ki az adott dátumok értékeit, ehhez könnyedén találhatunk számolásokat.

Ezekről a számolásokról viszont elmondhatjuk, hogy nem adnak valós értéket, különböző tényezők miatt hónapokat tévedhetnek, így kis utánajárással kideríthetjük, hogy a jelenleg kiszámolt mozgó ünnepek dátumai egyrészt számoláson másrészt meg egyezésen alapulnak.

Kimondhatjuk tehát, hogy saját számolások elvégzése, jelen esetben nem megvalósítható.

4.4.3. Félautomata megoldás

A végleges megoldás egy részben manuális karbantartást igénylő implementáció, melynek során az állandó dátummal rendelkező dátumok első beállítást követően újra számolódnak, a nem változó dátum jelölés esetén viszont az adminisztrátorra bízuk a konfigurációt.

4.5. Folyamatos Integráció beállítása Gitlabon

A már említett CI/CD folyamatok beállítása nagy segítséget tud nyújtani és a fejlesztést is meggyorsíthatja, a kódbázis felépítése és a tesztek futtatása idő és erőforrás igényes lehet a saját számítógépünkön, így ezt a munkát delegálva időt nyerhetünk.

A gitlab verziókezelő szolgáltatás lehetőséget nyújt saját Folyamatos integrációs csővezeték definiálásra amiben, az építés, a tesztek futtatása és a generált adatok alapján a tesztlefedettséget mérő munkameneteket könnyedén definiálhatjuk.

4.5.1. .gitlab-ci.yml konfiguráció

A gitlab szolgáltatásának részét képezi az alap szintű integrációs folyamatok létrehozása, ezek indulása automatikus. Nekünk csupán egy konfigurációs .yml kiterjesztésű fileban kell definiálnunk az általunk futtatni kívánt folyamatokat és az azokhoz tartozó esetleges környezeti változókat, vagy parancsokat.

A stages érték megadása után definiálnunk kell a munkamenetek nevét és sorrendjét.

```
stages:
  - build
  - test
  - coverage
```

Ezután konfiguráljuk a munkamenet, úgy hogy hivatkozunk a nevére majd megadjuk, hogy milyen docker image környezetben fusson az adott folyamat.

```
build:
  stage: build
  image: maven:3.6.3-jdk-11
```

Legvégül a script érték alatt felsoroljuk milyen parancsokat hajtson végre a docker imagen belül.

```
script:
  - mvn $MAVEN_CLI_OPTS compile
```

4.5.2. További felhasználható funkció

Ez a gyors konfiguráció nem csak a fejlesztést gyorsíthatja fel de akár Folyamatos szállítást is definiálhatunk vele, egy olyan munkamenet segítségével ami a felépített alkalmazásunk futtatható .war vagy .jar kiterjesztésű futtatható elemeit becsomagolja egy docker konténerbe, ssh kapcsolatot létesít egy általunk definiált IP címmel és ott a megadott helyre kitelepíti az alkalmazásunkat.

Ezzel elérhetjük, hogy a felhasználó a nap végére akár több hibajavítás eredményét vagy akár pár hetente egy új funkció használatát is élvezhesse.

5. fejezet

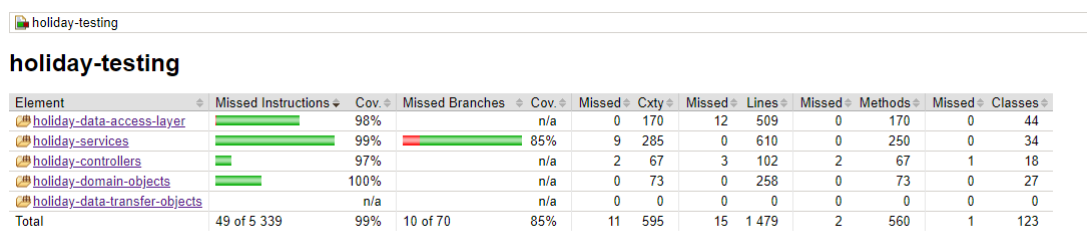
Rest applikáció tesztelése

5.1. Unit tesztek

A fejlesztés során folyamatos logikai egység tesztek írása már fejlesztési időben jelentős mennyiségű tervezési és megvalósítással kapcsolatos hibára mutat rá.

A megvalósítás részeként a konfigurációs és kizárólag nyelvi elemeket tartalmazó osztályok kivételével, folyamatos tesztlefedettség ellenőrzéssel történt az implementáció.

A kódbázis, összesen 312 db egység tesztet tartalmaz, amelyek a mérések alapján 99%-os tesztlefedettséget jelentenek. Mint ahogy azt az 5.1 ábra is mutatja.



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
holiday-data-access-layer	<div></div>	98%	<div></div>	n/a	0	170	12	509	0	170	0	44
holiday-services	<div></div>	99%	<div></div>	85%	9	285	0	610	0	250	0	34
holiday-controllers	<div></div>	97%	<div></div>	n/a	2	67	3	102	2	67	1	18
holiday-domain-objects	<div></div>	100%	<div></div>	n/a	0	73	0	258	0	73	0	27
holiday-data-transfer-objects	<div></div>	n/a	<div></div>	n/a	0	0	0	0	0	0	0	0
Total	49 of 5 339	99%	10 of 70	85%	11	595	15	1 479	2	560	1	123

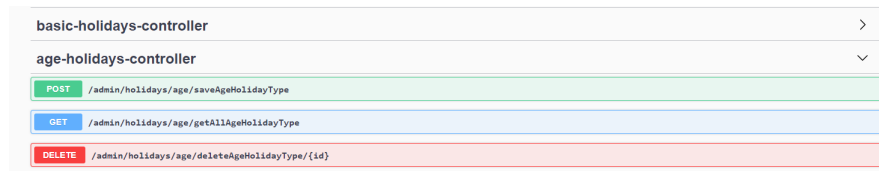
5.1. ábra. Jacoco riport generált adatai.

Az alkalmazás teljes felépítése a tesztek futtatásával átlagosan 40 és 45 másodperc közé tehető.

5.2. Funkcionális tesztek Swagger UI segítségével

A WEB applikációk szerver oldali logikája megjelenítési rétegtől függetlenül tesztelhetők, az általunk létrehozott végpontok erre alkalmas programok segítségével és a megfelelő HTTP kérésekkel meghívhatóak és a program futási időben kiszolgálja azokat.

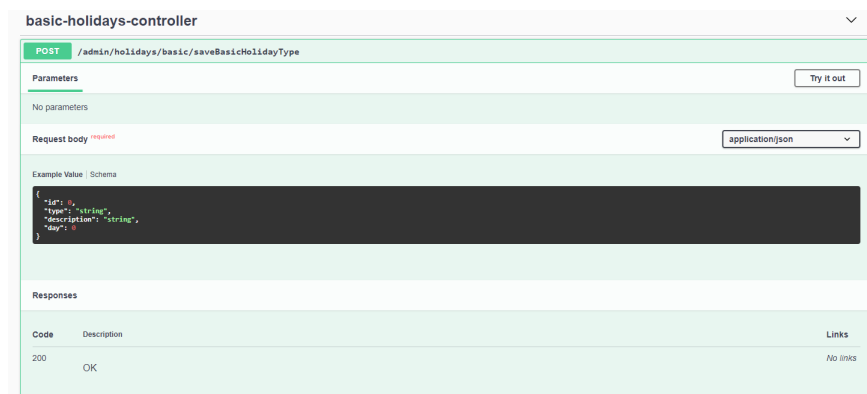
Ahhoz, hogy ne kelljen külső programot használni egy külső könyvtár segítségével saját felületet generálhatunk az végpontjaink számára. A kontextus épülésekor a külső függőség meghívódik és az általunk vezérlő típusként definiált osztályok HTTP hívással megjelölt függvényei alapján megjelenítést generál ami jól látható az 5.2.



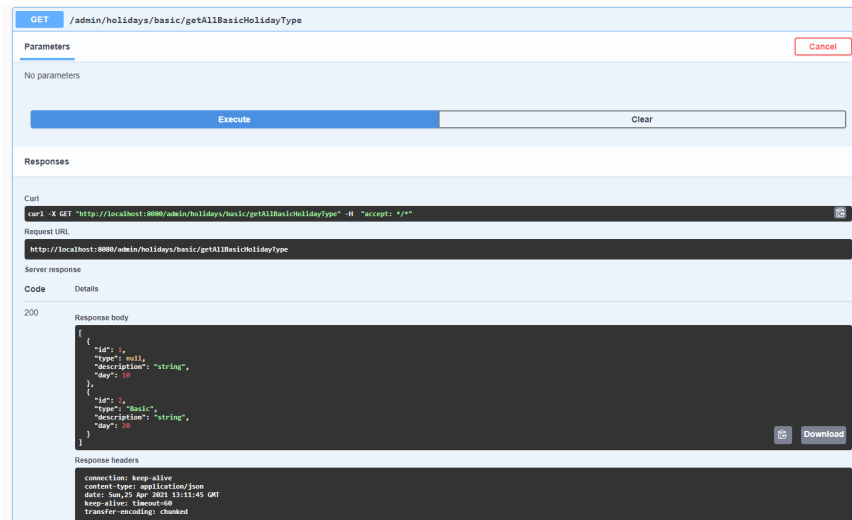
5.2. ábra. Swagger UI felület.

Előnye, hogy nem csak láthatóvá teszi végpontokat, de dokumentációként is szolgál számunkra a bemeneti és visszatérési értékekről egyaránt. Továbbá biztosít még egy kitölthető űrlapot és egy mintát a aminek a segítségével a megfelelő JSON hívással ki is próbálhatjuk az adott funkció működését.

A küldés az 5.3 ábrán az 5.4 pedig a lekérdezés tesztelése látható.



5.3. ábra. Swagger UI Post hívás tesztelése.



5.4. ábra. Swagger UI Get hívás tesztelése.

6. fejezet

Összefoglalás

A szakdolgozatom elkészítését nagyban segítette, a Java Enterprise környezetben elsajátított szakmai tapasztalatom a webalkamázásokkal kapcsolatban. A kutatómunka, és a téma megismeréséhez szükséges előkészületek igen hasznos tudást nyújtottak a munkajog terén, valamint a komolyabb szoftvertervezéssel kapcsolatos ismereteimet is sikerült bővítenem. Az implementáció hosszas folyamata során kifejezetten sok új és hasznos szakmai ismeretanyagra tettem szert.

A program jelenlegi állapotában működőképes, de a teljes funkciókövetelménynek nem sikerült eleget tenni, így bőven tartalmaz még magában fejlesztési lehetőséget.

6.1. Fejlesztési lehetőségek

Mivel több követelmény nem teljesült ezért több lehetőséget is szeretnék kifejteni, de mindezek előtt a továbbfejlesztésben az architektúra újragondolását tartom a legnagyobb lehetőségnek.

Architektúra szempontjából, a microservice irányában folytatnám a program fejlesztését, melyet az adatbázis, és a megjelenítési réteg önállósításával lehet elkezdni. így a központi logika összetettsége csökkenne, és a kódbázisok külön fejleszthetők lennének.

A Statisztika generálás funkcióra külön alkalmazást építenék, amely nem az összetett szabadságot foglalkozó webalkalmazás egy modulja, hanem egy önállóan fejleszthető és karbantartható egység.

A szoftverbiztonság szempontjából, a Spring által támogatott és egyetemeken is alkalmazott LDAP integráció fejlesztése. Ezzel a fejlesztéssel felhasználó szempontból megkönnyítjük a használatot, adatbázisunk csak és kizárólag a számunkra hasznos adatokat tárolná.

Kérelem jóváhagyás esetén a már említett Flowable könyvtár használatával komolyabb utánaajrást követően lefejleszthető.

A felsorolt fejlesztési lehetőségeken mellett a már specifikált, de még nem megvalósított részleteket mint például az éves terv generálása, valamint a megjelenítési réteg teljes implementálása természetesen előnyt élvez a fejlesztési sorban.

Irodalomjegyzék

- [1] Flowable AG. Flowable open source documentation.
<https://flowable.com/open-source/docs/form/ch02-Configuration/>, 2020.
- [2] The Project Lombok Authors. Project lombok.
<https://projectlombok.org/>, 2009.
- [3] Adorján Botond. Adatbázisok verziókövetése liquibase és liquiface segítségével.
<http://weblabor.hu/cikkek/adatbazisok-verziokovetese>, 2013.
- [4] AnnualLeave Company. Annualleave weboldal.
<https://annualleave.com/>.
- [5] Leave Dates Company. leavedates weboldal.
<https://www.leavedates.com/>.
- [6] dr. Babati Szabolcs. A munka törvénykönyve szabadságra vonatkozó rendelkezései.
<https://www.kormanyhivatal.hu/download/1/af/60000/A%20munka%20t%C3%B6rv%C3%A9ny%C3%B6nyve%20szabads%C3%A1gra%20vonatk%C3%B3z%C3%A9sei.pdf>, 2013.
- [7] dr. Kovács Szabolcs. A szabadság számítása tört év esetén.
<https://ado.hu/munkaugyek/a-szabadsag-szamitasa-tort-ev-eseten/>, 2016.
- [8] dr. Kozma-Fecske Ivett. Mit kell tudni a fizetés nélküli szabadságról?
<https://ado.hu/munkaugyek/mit-kell-tudni-a-fizetes-nelkuli-szabadsagrol-2>, 2020.
- [9] excelmester. Szabadság tervező nyilvántartó excel táblázat program 1.
<https://excelmester.hu/k-excel-tablazat-letoltesek/szabadsag-tervezo-nyilvantarto-excel-tablazat-program-1/>.
- [10] F.P. Fizetett szabadság 2021. év szabályai, kiadása.
<https://www.mosthallottam.hu/hasznos-info/fizetett-szabadsag-2021-ev-szabalyai-kiadasa/>, 2020.
- [11] freshworks Company. freshworks, freshteam weboldal.
<https://www.freshworks.com/hrms/>.
- [12] Ismeretlen. Ci/cd.
<https://hu.wikipedia.org/wiki/CI/CD>, 2019.

- [13] Ismeretlen. Objektum-relációs leképzés.
https://hu.wikipedia.org/wiki/Objektum-rel%C3%A1ci%C3%B3s_lek%C3%A9p%C3%A9s, 2019.
- [14] Ismeretlen. Apache subversion.
https://hu.wikipedia.org/wiki/Apache_Subversion, 2020.
- [15] Ismeretlen. Egységtesztelés.
<https://hu.wikipedia.org/wiki/Egys%C3%A9gtesztel%C3%A9s>, 2020.
- [16] Ismeretlen. Git.
<https://hu.wikipedia.org/wiki/Git>, 2020.
- [17] Ismeretlen. Spring keretrendszer.
https://hu.wikipedia.org/wiki/Spring_keretrendszer, 2020.
- [18] Ismeretlen. Apache maven.
https://hu.wikipedia.org/wiki/Apache_Maven, 2021.
- [19] Ismeretlen. Java (programozási nyelv).
[https://hu.wikipedia.org/wiki/Java_\(programoz%C3%A1si_nyelv\)](https://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv)), 2021.
- [20] Ismeretlen. A kontroll megfordítása.
https://hu.wikipedia.org/wiki/A_kontroll_megford%C3%ADt%C3%A1sa, 2021.
- [21] Ismeretlen. Modell-nézet-vezérlő.
<https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91>, 2021.
- [22] Ismeretlen. Verziókezelés.
<https://hu.wikipedia.org/wiki/Verzi%C3%B3kezel%C3%A9s>, 2021.
- [23] Evolution Consulting Kft. Hrmaster.
<https://hrmaster.hu/termek>.
- [24] Leviathan Solutions Kft. Predor.
<https://www.predor.hu/Home/Megoldasaink>.
- [25] Magyarország kormánya. 1992. évi xxxiii. törvény a közalkalmazottak jogállásáról.
<https://net.jogtar.hu/jogszabaly?docid=99200033.tv>, 1992.
- [26] Dr. Pál Lajos. A szabadság elszámolásának egyes kérdései.
<https://munkajogilap.hu/a-szabadsag-elszamolasanak-egykes-kerdesei/>, 2018.
- [27] Robert C. Martin. *Clean Code : A Handbook of Agile Software Craftsmanship*. Pearson Education (US), Upper Saddle River, NJ, United States, 2009.
- [28] Dr. Faragó Máté. A szabadság kiadásának szabályai.
<http://www.mkpartners.eu/hu/?mod=news&cla=news&fun=shownews&id=107&temp=base>, 2019.

- [29] Calamari sp. z o.o. sp. k. Calamari weboldal.
<https://calamari.io/leave-management>.
- [30] Till Zoltán. Szabadság tervező és nyilvántartó.
http://www.hatekonysag.hu/hr_szabadsagok.php.

Adathordozó használati útmutató

Ez jellemzően csak egy fél-egy oldalas leírás. Arra szolgál, hogy ha valaki kézhez kapja a szakdolgozathoz tartozó CD-t, akkor tudja, hogy mi hol van rajta. Jellemzően elég csak felsorolni, hogy milyen jegyzékek vannak, és azokban mi található. Az elkészített programok telepítéséhez, futtatásához tartozó instrukciók kerülhetnek ide.

A CD lemez tartalma

- a `dolgozat.pdf` fájl,
- a `dolgozat_latex` mappa tartalmazza a LaTeX forráskódját a dolgozatnak,
- a `program` mappa tartalmazza az elkészített program forráskódját

A program futtatásához a következő előfeltételek szükségesek:

- A futtató számítógépre telepített Docker környezet
- Legalább Java 11-es verzió.
- Configurált Maven
- Node.js
- Angular.js

Az előfeltétel teljesítése után a következő parancsokkal indítható a program.

- a `holiday_registry` könyvtárban állva console-ban kiadjuk az **`mvn clean install`** parancsot
- a Frontend demo indításához a `frontend` mappában consoleon keresztül kiadjuk az **`ng serve`** parancsot (Ezután a `localhost:4000` portján elérhető az oldal)
- a `docker` könyvtárban consoleból kiadjuk a **`docker-compose up`** parancsot.
- a Backend applikáció futásához a `web` könyvtárból kiadjuk az **`mvn spring-boot:run`** parancsot a program felülete a `http://localhost:8080/swagger-ui/` linken elérhető.