Szegedi Tudományegyetem
Informatikai Intézet

GÉZA – Informatikus zárthelyi dolgozat ütemező és menedzselő rendszer

Szakdolgozat

Készítette:
Vad Avar
Mérnökinformatikus BSc
szakos hallgató

Témavezető:
Dr. Németh Gábor
adjunktus

Szeged
2025

Feladatkiírás

A témát kiíró oktató neve: Németh Gábor

A témát meghirdető tanszék: Képfeldolgozás és Számítógépes Grafika Tanszék

Típus: Szakdolgozat

Ki jelentkezhet: 1 fő, Mérnökinformatikus BSc szakos hallgató

A feladat rövid leírása:
A technikai korlátozást használó zárthelyi dolgozatok adminisztrációja meglehetősen komoly odafigyelést jelent az Irinyi Kabinet üzemeltetőinek. Ráadásul az oktatók számára nehezen ellenőrizhető, hogy igényeik megfelelően kerültek-e beállításra. Ezt a problémát hivatott megoldani a fejlesztendő rendszer, ahol az oktatók elküldhetik igényeiket a technikai korlátozásokat illetően a Kabinet üzemeltetőinek, akik a beállításokat vissza tudják jelezni a rendszerben. Az oktatók naptárnézetben láthatják a termek és a kurzusok beállításait. A felhasználók az intézeti LDAP bejelentkezéssel érhetik el a rendszert. Az órarendi adatokat a rendszernek a BIR2 rendszer adatbázisából kell kiolvasnia. A rendszert webes technológiákkal kell megvalósítani.

Előismeretek: ajánlott ismerni a zárttermi dolgozatok technikai korlátozásának menetét

Szakirodalom: főként angol nyelvű

Tartalmi összefoglaló

- ***A téma megnevezése:***

Géptermi technikai korlátozásokat ütemező és menedzselő rendszer, az Irinyi Kabinet részére.

- ***A megadott feladat megfogalmazása:***

A Szegedi Tudományegyetem Informatikai Intézetének Irinyi Kabinetjében a technikai korlátozásos zárthelyi dolgozatok adminisztrációja jelentős odafigyelést igényel, és az oktatók számára nehezen ellenőrizhető az igények helyes beállítása. A feladat egy webes rendszer kifejlesztése volt, amely elektronikus úton teszi lehetővé az oktatók számára igényeik rögzítését, az üzemeltetők pedig átlátható módon kezelhetik ezeket. A rendszer célja a hibalehetőségek minimalizálása és az oktatói-operátori munkafolyamatok egyszerűsítése az intézeti LDAP és a BIR órarendkezelő rendszer integrálásával.

- ***A megoldási mód:***

Modern webalkalmazás készült React és NextJS technológiákkal. A rendszer LDAP-alapú autentikációt használ Better-Auth keretrendszerrel, MySQL adatbázist Drizzle ORM közvetítésével, és közvetlenül kapcsolódik a BIR órarendi adataihoz. Az oktatók csak saját kurzusaikhoz foglalhatnak. A megoldás szerepkör-alapú jogosultságkezelést valósít meg, automatikus e-mail értesítéseket küld, és modulárisan bővíthető struktúrát követ.

- ***Alkalmazott eszközök, módszerek:***

React 19, NextJS 15, TypeScript, TailwindCSS, ShadCN komponenskönyvtár a frontend oldalon. Backend: NextJS szerveroldali függvények, Better-Auth LDAP autentikációval, MySQL adatbázis, Drizzle ORM. Külső integrációk: BIR adatbázis, SMTP levelezés. Git verziókezeléssel fejlesztve.

- ***Elért eredmények:***

A rendszer csökkenti a hibalehetőségeket strukturált, validált foglalási folyamattal. A BIR integráció biztosítja, hogy oktatók csak érvényes kurzusaikhoz foglalhatnak. Az operátorok központi felületen kezelik az igényeket. Automatikus e-mail értesítések tájékoztatják mindkét felet. A naptárnézet átláthatóvá teszi a Kabinet foglaltságát. A moduláris architektúra alapot teremt a jövőbeli automatizáláshoz, miközben megőrzi az operátori kontrollt.

- ***Kulcsszavak:***

webalkalmazás fejlesztés, LDAP autentikáció, NextJS, React, adatbázis tervezés, zárthelyi adminisztráció, erőforrás-ütemezés, jogosultságkezelés

Tartalomjegyzék

BEVEZETÉS

Az Informatika Intézetben jelenleg dolgozó oktatók sokszor találkozhattak már a zárhelyi dolgozat íratás problémájával és az ezzel járó adminisztratív teendőkkel. Ezek közül néhányat, a teljesség igénye nélkül, kiemelve: dolgozat összeállítása, feladatok kitűzése, kitűzött feladatok ellenőrzése, pontozás kitalálása. A felsoroltokon kívül gondolkodni kell azon, hogy a dolgozat megírásához egy olyan környezetet kell biztosítani, amelyben a hallgatóknak minden rendelkezésre áll ahhoz, hogy a tudását tudja bizonyítani. Oktatói szempontból viszont ne legyen túl nagy feladat, bizonyos esetekben akár 60 hallgató, egyszeri felügyelete és a számonkérés tisztességességének biztosítása.

Ezen feladat megkönnyítésére az Irinyi Kabinetben rendelkezésre állnak olyan eszközök, amelyek arra hivatottak, hogy dolgozatok során az oktatónak ne kelljen 60 ember mögött ott állnia annak érdekében, hogy biztosítsa a korrekt eredményeket. Ezen eszközcsoportot a hallgatók és oktatók egyaránt „ZH mód” vagy „vizsga mód” néven ismerik. Az egyszerűség kedvéért ezen dolgozat hivatkozik erre az eszközre ezeken a neveken is.

Ez a fajta üzemmód lehetővé teszi azt, hogy a kabineti számítógépeket távolról vezérelve bizonyos funkciókat (pl. internetelérés, szoftverek használata, operációs rendszerek stb.) a rendszert üzemeltető operátorok letiltsanak, ezzel segítve a „steril” környezet biztosítását.

A fent említett adminisztratív elfoglaltságok közé tartozik az is, hogy ennek az üzemmódnak a beütemezését kérje az oktató a saját órájára. Ez a kérés jelenleg egy ímélváltást jelent az oktató és az üzemeltetés között. Ezen kérést elolvasva az operátor beállítja az oktató által kért beállításokat. Így leírva ez a fajta, évek alatt már megszokott, protokoll nagyon jól működik, elméletben. A gyakorlat viszont mást mutat.

Látható, hogy sok a lehetőség hibázásra, mivel a folyamat nagyrészét emberek végzik. Ha az oktató elgépel valami a kérésben, megtörténhet, hogy rossz beállítást kapnak a hallgatók, ha az operátor gépel el valamit, előfordulhat, hogy nem is jó teremben indul el a ZH mód.

A dolgozat pedig pont ezt a problémát tűzte ki maga elé, létre kell hozni egy rendszert, amely hosszútávon képes minimalizálni vagy teljesen kiküszöbölni a hibák lehetőségét. Úgy, hogy közben megmarad az operátori rálátás a teljes rendszerre, és az oktatók pedig továbbra is képesek zárhelyi dolgozat közben olyan környezetet biztosítani a hallgatóknak, amely lefed minden felmerülő igény, mind hallgatói mind pedig oktatói oldalról.

Ezen dolgozat keretében elkészült szoftver alapja lehet egy ilyen rendszernek, viszont várhatóan finomításra és továbbfejlesztésre szorul majd. Addig is amíg ezen módosítások

megtörténnek a program minimalizálja a jelenlegi hibalehetőségeket és teret ad automatizációk integrálásának.

Az alábbi fejezetek sorra a következő témaköröket dolgozzák fel: Először áttekintjük, hogy mi is az a vizsga mód, hogyan működik milyen szabályok vannak, azokat hogyan lehet alkalmazni és milyen gyengeségek figyelhetőek ebben meg. Utána egy távoli áttekintést adunk arról, hogy az elkészült rendszer pontosan micsoda, és hogyan próbálja megcélozni a jelenlegi szisztéma gyengeségeit. Ezután mélyebbre ásunk az „új kolléga” megvalósításában, áttekintve a háttérben húzódó technológiákat. Ezt követően az adatstruktúrát tekintjük át. Majd architekturális szinten vizsgáljuk meg az elkészült szoftvert, kitekintve arra, hogy a jogosultságkezelés hogyan van megoldva. Ezentúl röviden megvizsgáljuk, hogy milyen integrációs lehetőségek vannak, amelyek biztosítják, hogy valóban használható legyen a program. Végül pedig egy jövőbeli kitekintéssel zárunk, részletezve, hogy milyen potenciál van az alkalmazásban hosszútávon.

1. A VIZSGA MÓD

Az Irinyi Kabinet géptermeiben jelenleg „ZH mód” vagy „vizsga mód” néven ismert speciális üzemmód hivatott néhány alapvető technikai korlátozást bevezetni a hallgatói számítógépeken a számonkérések alatt. A Kabinetben erre több eszköz is rendelkezésre áll, ezen eszközök összességét nevezzük ZH módnak, esetenként pedig vizsga módnak.

A termekben lévő számítógépeken, a felhasználók által látható operációs rendszer valójában egy virtuális rendszer [1]. Ezt a felhasználó egy, a hoszt gépen futó, menüből tudja saját magának kiválasztani mikor használni szeretné a számítógépet. Ez az első pont, ahol a vizsga mód egyik eszköze bele tud nyúlni a hallgató által már megszokott folyamatba. Lehetőség van ugyanis távolról befolyásolni, hogy milyen választható opciók jelennek meg ebben a menüben. Így, ha szeretné az oktató, hogy egy dolgozat csak Linux rendszerről legyen megoldható, akkor képes a többi választási lehetőséget innen eltüntettetni.

Ez a virtualizált megoldás lehetőség ad olyan rendszerképfájl betöltésére, amely az általános menüben nincs benne. Így biztosítva azt, hogy ha szükséges akkor más szoftvereket, szoftverbeállításokat kapjanak a gépek bekapcsoláskor. Ezen megoldás arra is képes, hogy az elindult gépek mindegyikén rajta legyen egy feladatsor, illetve kiindulási fájlok a dolgozat megkezdésekor.

A virtualizáció viszont kétélű fegyver. A fenti pozitív hatásai mellett megemlítendő, hogy a virtuális rendszer kikapcsolásakor a rajta lévő adatok is eltűnnek (megjegyzés: ez konfiguráció kérdése, a dolgozat a Kabinetben tapasztalható működést írja le). Ez adatvédelmi

szempontból rendkívül hasznos, nem kell azzal foglalkozni, hogy valaki esetleg bejelentkezve maradt, és a következő hallgató pedig kihasználva ezt a nevében cselekszik. Az érme másik oldalán pedig ott van, hogy egy zárhelyi megírása alatt nem szerencsés eset az, ha valami miatt a számítógépet újra kell indítani (ilyen eset lehet például egy technikai hiba vagy áramkimaradás) akkor a hallgató munkája teljesen elveszik. Ez a probléma a kiküszöbölhető azzal, ha valamilyen online felületre kell feltölteni a hallgatóknak a megoldásaikat, ahonnan esetlegesen vissza is tudják azt tölteni újraindítás után. Viszont vannak olyan esetek, mikor ez nem megoldható vagy túlzott, ok nélküli, többletmunkát okozna. A Kabinetben van erre kidolgozott eszköz, amely képes megoldást kínálni erre. Ezt a „Saját könyvtár használata (home)” fejezet tárgyalja.

Hálózati szempontból vizsgálva a termek kiépítését az észrevehető, hogy minden terem egy saját virtuális hálózatban van (VLAN) (Angyal Tamás, személyes egyeztetés, 2025. március 6.). Ez lehetővé teszi egy újabb eszköz bevetését, amellyel távolról vezérelhető, hogy a számítógépek hogyan érik el az internetet. Ezt úgy kell elképzelni, hogy az üzemmód bekapcsolásakor teremspecifikusan, azaz a terem virtuális hálózatára értve, vezérelhető, hogy pontosan milyen internetes címeket érnek el az eszközök. Így biztosítható, hogy a hallgatói számítógépek semmilyen módon ne férjenek hozzá az internethez, ha erre valamelyik oktató igényt tart. Kizárva ezzel a külső kommunikációs csatornákat vagy segítő eszközöket.

1.1. Szabály jelentése, alkalmazása

Ezen dolgozatban, ha ZH mód kontextusában, szabályra hivatkozunk, akkor egy olyan beállításösszességre referálunk, amelyek egyszerre, együttesen vannak alkalmazva egyetlen terem számítógépeire.

Az üzemeltetéssel folytatott egyeztetések alapján a jelenlegi rendszerben 2, 3 vagy 4, az előző fejezetben leírt beállítás alkot egy szabályt:

- rendszerválasztó menüben elérhető opciók,
- internet- és hálózatkorlátozások,
- választható módon: ZH könyvtár a hallgatóknak,
- választható módon: használt operációs rendszer leállítása első és/vagy utolsó lépésként.

Amióta a jelenlegi architektúrát használja a Kabinet összegyűltek már jól bevált, tesztelt és valamennyire moduláris szabályok. Amennyiben ezek közül kell valamelyiket alkalmazni, az operátori feladat lényegében annyi, hogy a megfelelő helyre be kell illeszteni a megfelelő fájl elindítására szolgáló sort.

A szabályok alkalmazását legegyszerűbb oly módon áttekinteni, hogy egy konkrét példát veszünk, és a lépésenként vizsgáljuk. Feltételezzük, hogy a kérés, egy *Programozás Alapjai Gyakorlat* tárgyhoz érkezett, és a kért szabály már létezik, azaz módosítás nélkül csak be kell állítani a megfelelő kurzus időpontjára. A szabály, amit alkalmazni kell a következő beállításokat tartalmazza: a számítógép a kezdéskor újraindul, csak Linux operációs rendszer indítható el, a dolgozat ideje alatt csak a Bíró3 [2] feladatbeadó és -értékelő oldal érhető el az interneten, használhatóak a JetBrains [3] fejlesztőeszközei, a számokérés végén nem indul újra a számítógép, viszont az internet újra elérhető lesz a megszokott módon.

Első lépésként egy belső, erre a célra kijelölt szervert kell elérnie az operátornak, amelynek az időzített feladati közé (cron) fel kell venni egy újat a kért időpontra, mely elindít egy parancsfájlt („scriptet”).

Mikor elérkezik a kezdési időpont, elindul az állomány, mely valamilyen sorrendben elvégzi a beállításokat. Mivel egy *Programozás Alapjai* dolgozathoz kérték a beállításokat az alább kifejtett események történnek.

A kérésnek megfelelően a jelenleg használt operációs rendszer leáll és újra el kell azt indítani – tehát előre megnyitott weboldalak/programok bezáródnak. A választómenüben csak a Linux operációs rendszer lesz elérhető. A többi választható opció eltűnik.

A terem virtuális hálózatára olyan *access-control list* (ACL) [4] beállítások kerülnek alkalmazásra, amelyek tiltják az internet általános elérését, és csak a Bíró3 [2] érhető el. Legalábbis felhasználói szemmel ez látszik, valójában több dolog is elérhető az internetről, amelyek szükségesek ahhoz, hogy a használt szoftverek valójában működjenek. Ennek viszont felhasználói szempontból nincs jelentősége, mert az, hogy egy licenszszerver elérhető alapfeltételezés a használónak. Illetve természetesen a Bíró3 [2] sem egy olyan alkalmazás, amely önmagában áll, vannak külső függőségei, amelyek szintén át vannak engedve ezen a szűrőn.

Ezen felül az előre létre hozott „ZH home” könyvtárak lesznek felcsatolhatóak a „Home Mount” előre megírt script futtatásával. Ez biztosítja, hogy lehetőség van a Bíró3 [2] felületén kívül is adatmentésre a felmérés alatt. Ezen fájlok csak a dolgozat ideje alatt lesznek elérhetők, utána hallgatóként nem lehet hozzáférni.

Ebben az esetben mind a négy lépést elvégezte a szabály beállításához a rendszer. Az újraindítás módja az választható, szabályonként eltér, ugyanúgy ahogy a saját könyvtár létezése is.

1.2. Saját könyvtár használata (home)

Az Intézet mindenki számára biztosít egy saját könyvtárat, amelyet a hallgató alapesetben bármikor bárhonnan elérhet, és tárolhat rajta fájlokat. Ez a könyvtár elérhető bárhonnan az internetről, csak a felhasználó nevet (ismertebb nevén a „h-s azonosítót") és a hozzá tartozó jelszót kell tudni. Egy ilyen könyvtár azonban lehetőséget kínál azoknak, akik külső segítséget szeretnének használni egy dolgozat alatt.

Nyilvánvaló, hogy nem lenne szerencsés, ha ehhez a könyvtár hozzáférhető lenne a dolgozat megírása közben, ugyanis ebben elmenthető bármilyen állomány, akár otthonról, ami aztán a számonkérés alkalmával letölthető és használható. Éppen emiatt a ZH üzemmódba állított számítógépek egy külön erre a célra létrehozott home könyvtárat csatolnak fel. Ez a könyvtár kezdetben üres, a dolgozat ideje alatt a hallgató hozzáférhet, menthet bele fájlokat, és ha újra kell indítani a számítógépét, ezek akkor is megmaradnak. A felmérés után pedig, kérés esetén, ezen fájlokat az oktató megkaphatja, és szükség esetén átnézheti, kiértékelheti azokat.

Ez a megoldás lehetővé teszi az üzemeltetőknek, hogy teljesen leválasszák a hallgatókat az internetről, ha szükséges. Azon probléma fennállása nélkül, hogy nem tudják a hallgatók lementeni a munkájukat, és valamilyen technikai probléma miatt elveszne minden, amin dolgoztak. Technikai oldalról ennek a pontos megvalósítása nem lényeges jelen kontextusban.

1.3. Feladat kiosztása minden gépre

Az utóbbi néhány évben talán már egy kevésbé jellemző funkciója a vizsgamódnak az, hogy az oktatónak lehetősége van fájlokat feltölteni a számítógépekre, így mikor azok elindulnak a hallgatóknak már rendelkezésre állnak feladatok vagy kiindulóállományok.

Ezt az eszközt leváltotta jórészt a CooSpace [5], illetve az ahhoz hasonló egyéb online, vagy belső hálózatról elérhető szolgáltatások. Példaképp a Bíró3 [2] használatához nem szükséges ezt a funkciót használni, ugyanis az a feladatbeadási felület mellett magával a feladattal is rendelkezik. Így fölösleges és csak további terhelés (potenciális hibaforrás) az, hogy a feladatok külön állományként is felkerüljenek a gépekre.

A múltban viszont ez használt eszköz volt ez, és mivel a Kabinet alapvető architektúrája azóta nem változott szignifikánsan, ezért ez továbbra is egy igénybe vehető lehetőség.

1.4. A rendszer jelenlegi gyengeségei

A fenti, igencsak komplex, rendszert áttekintve látható, hogy még akkor is, ha feltételezzük, hogy az évek óta megírt és használt beállítások, scriptek nem hibásodnak meg, van néhány olyan pont, ahol az emberi beavatkozás miatt hibák léphetnek fel.

Az első ilyen hibaforrás rögtön a levél, amit az oktató ír az üzemeltetésnek, amennyiben abban az időpontot elrontja, vagy rossz szabályt kér akkor elképzelhető, hogy a számonkérés nem úgy fog történni, ahogy elképzelte. Azt lehet feltételezni, hogy amennyiben rossz teremszámot, vagy rossz idősávot (a hét napja, napon belüli óra) ad meg azt a túloldalon ellenőrzés után az operátor elutasítja, ugyanis az üzemeltetők beállítás előtt minden esetben ellenőrzik az órarendet is, manuálisan.

Természetesen az operátor is hibázhat, viszont ebben az esetben nagyobb problémák is adódhatnak. Egy rosszul beírt dátum, teremszám vagy időpont jelentheti azt, hogy az eredetileg igényelt ZH mód nem kapcsol be, viszont egy másik teremben, ahol elképzelhető, hogy éppen oktatás zajlik, pedig újra indulnak a számítógépek. Szerencsére ezen hibák ritkán fordulnak elő, a lehetőség viszont megvan rá.

Tegyük fel, hogy hiba nélkül sikerült beállítani mindent. Az oktató az igényét elküldte, az alapján az operátor helyesen állította be az időzítéseket és a megfelelő scriptet használat. Ám előre nem látható okok miatt változtatni kell valamelyik beállításon. Egy elképzelt forgatókönyv lehet, hogy a kiadott feladatsorban hiba van, ami miatt az oktató úgy dönt, hogy extra időt ad a hallgatóknak a megírásra. Mivel viszont a ZH mód fix időre van beállítva, ezért annak végeztével – beállítástól függően – kikapcsolhatnak a számítógépek. Ami azt jelenti, hogy a hallgató azon munkája, amit nem a home könyvtárában tárolt, vagy nem töltött fel egy online beadófelületre, elveszik. Ezt viszont el lehet kerülni azzal, ha időben jelzi az oktató a problémát az üzemeltetés felé. Ekkor ugyanis van lehetőség arra, hogy meghosszabbítsák az üzemmódot a teremben. Viszont ez nem minden esetben kivitelezhető megoldás, különböző okok miatt (pl. felügyelet nélkül hagyott terem, szünetét töltő operátor, utolsó pillanatos változtatás).

Az előző eset feltételezte, hogy a kommunikáció a felhasználó és az operátor között lezajlott. Néhány esetben viszont ez nem egy ímélváltás oda-vissza, hanem levelezések hosszú sora, amit egy idő után nehéz követni. Ez pedig előidézheti az alfejezet elején tárgyalt hibákat.

2. GÉPTERMI ZH ADMINISZTRÁTOR (GÉZA)

Az előző fejezetben említett gyengeségeket felismerve és azok hatásának csökkentésére jött létre a „Géptermi ZH Adminisztrátor” röviden csak GÉZA néven fejlesztett alkalmazás. Bizonyos szempontból tekinthető új üzemeltetési kollégának is. GÉZA egy olyan webalkalmazás, amely megkönnyíti mind operátori, mind oktatói szempontból a jelenlegi vizsgamód rendszer valamennyi aspektusát.

Oktatói szempontból egy felhasználóbarát, modern felületet kínál az igények benyújtására, amely beépített védelemmel rendelkezik a lehető legtöbb hibalehetőség ellen. Operátori szempontból pedig egy egységes, könnyen használható felületet kínál az igények kezelésére.

GÉZA az első lépése annak, hogy a vizsga mód rendszer jelenlegi robusztusságát megtartva megfeleljen az igencsak növekvő igényeknek.

2.1. GÉZA felépítése

GÉZA felépítésének tervezésekor fontos szempont volt, hogy a háttérrendszer, ami a tényleges vizsgamód beállításokat kezeli már adott. Úgy kellett erre ráépíteni, hogy a jelenlegi funkcionalitás megmaradjon, és emellett egy könnyen kezelhető, biztonságos és rugalmas alkalmazást kapjunk.

Ennek eléréséhez GÉZA alapját egy React [6] alapú keretrendszer szolgál, a NextJS [7]. A választás amiatt esett erre, mert a keretrendszer egyszerre kezeli a felhasználóoldali megjelenítéseket és a szerveroldali funkcionalitást is ellátja. Így fejlesztői szempontból könnyen átlátható, kezelhetővé teszi a projektet. Előnyei közé tartozik még az is, hogy egy hosszú ideje a piacon található könyvtár szolgáltatja az alapját, így hosszú távon nagy támogatottságra lehet számítani.

Fontos szempont volt, hogy a jelenleg rendelkezésre álló információkat ne kelljen egy további felületre is felvinni. Emiatt több integráció is része a teljes felépítésnek. Egyik ilyen például a beléptetés, amely a már megszokott *Lightweight Directory Access Protocol* (LDAP) [8] azonosítókat használja, másik példa erre pedig a *Belső Információs Rendszer* (BIR) [9], ami pedig az órarendi adatokat szolgáltatja [10].

2.2. Funkcionális követelmények

Az alábbi alfejezetekben megfogalmazott követelmények egy tipikus felhasználási forgatókönyvet végiggondolva és azt szem előtt tartva születtek meg. A megfogalmazásuk során az operátorokkal – mint egyik végifelhasználó –, és néhány oktatóval – mint másik

végifelhasználó – is történt egyeztetés, hogy a lehető legjobban le lehessen fedni az igényeiket.

2.2.1. LDAP bejelentkezés

Az internet egyre inkább eltávolodik a tipikus felhasználónév/ímélcím és jelszó párostól és egyre inkább a külső szolgáltatói azonosítás irányába mozdul a szabvány. Ez sokkal biztonságosabb és hosszú távon fenntarthatóbb rendszereket eredményez. [11, 12]

Így GÉZA tervezésénél alapvető követelmény volt, hogy saját azonosítási megoldás nélkül készüljön a rendszer. Az azonosítást teljes egészében a már létező, és az Intézeten belül használt, *Lightweight Directory Access Protocol* (LDAP) [8] biztosítja. Ezzel a beléptetés problémáját teljes egészében levéve GÉZA-ról. Az LDAP rendszerében mindenkinek saját, egyedi és központi azonosítója van, így belépve GÉZA számára már minden szükséges adat rendelkezésére áll, ami ahhoz kell, hogy biztosítsa a zökkenőmentes használatot.

Ez megakadályozza, hogy jogosulatlan felhasználók hozzáférjenek a rendszerhez, csökkenti a biztonsági réseket, és centralizálja a felhasználókezelést. Ez utóbbi igen nagy előny egy akkora szervezetnél, mint az Informatika Intézet, ahol közel sem lenne kivitelezhető, hogy minden egyes oktatónak a rendszerben külön felhasználót kelljen létrehozni.

2.2.2. Órarendkezelés

Az előbb említett egyedi LDAP azonosító lehetővé teszi azt, hogy egy másik, szintén gyakorlatban használt rendszert integráljunk be.

A *Belső Információs Rendszer* (BIR) [9] évek óta szolgáltat funkciókat az oktatóknak és hallgatóknak. Ezen funkciók között van az órarendkezelés is. Ide minden félév elején felkerülnek az aktuális kabineti órarendek – pontosan azok, amelyekre később ZH módot foglalhatnak majd az oktatók. Ezen adatokat pedig GÉZA fel tudja használni arra, hogy az oktatóknak egy olyan felületet adjon, amelyen kényelmesen és átláthatóan tudnak foglalni az órájukra vizsga módot.

Az integráció pontos részleteit a „Külső integrációk” fejezet taglalja, viszont, ami lényeges, hogy az adatokat egyenesen ezen felület adatbázisából veszi át GÉZA. Így elég ide felvinni az adatokat, és foglalásokat már létre is lehet hozni. Mivel a BIR [9] tárolja azt is, hogy az egyes órához mely oktatók vannak hozzárendelve, ezért ezt az információt is át tudjuk venni.

Az átvett órarendi információk biztosítják, hogy az oktatók csak saját órájukra, és csak azon belüli időpontokra tudjanak csak foglalni. Ezzel is limitálva a hibák lehetőségét.

2.2.3. Órarenden kívüli kérések

Nem lehet teljes egészében a BIR-re [9] hagyatkozni, ugyanis vannak olyan speciális esetek, amikor órarenden kívül is szükséges lehet ZH módot beállítani. Ilyen helyzetek tipikusan pót- és javítódolgozatok alkalmával vannak, esetleg vizsgaidőszakban, gyakorlati vizsgáknál vagy jelenlétet igénylő online vizsgáknál.

Ezekre is valamilyen módon lehetőséget kell biztosítania a rendszernek. Az egy kaotikus megoldás lenne, ha bárki (bármelyik oktató) tudna egyéni kéréseket létrehozni, bármikor amikor csak akar. Ezért egy olyan megoldást kellett megvalósítani, amely elérhetővé teszi ezt a funkciót, anélkül, hogy átláthatatlan lenne a rendszer. Ez a megoldás pedig az eredetileg is használt, ímél alapú kérések formájában valósul meg. Az operátor tud rögzíteni egyéni kérést bármikor, ha az oktató ezt ímélben igényli tőle.

2.2.4. Géptermek, mint erőforrások

A géptermekre tekinthetünk úgy, mint erőforrásokra, melyeket egyszerre csak egy felhasználó tud használni, azaz egyszerre csak egy ZH mód fut. Ezt a problémát alapvetően órarendtervezéskor megoldottnak tekintjük, azaz feltételezhetjük, hogy az órarend, amit rendszerünk megkap nem tartalmaz olyan helyzetet, amikor ugyanazon időpontban, két felhasználó is jogosult két különböző kurzushoz vizsgamódot foglalni, ugyanahhoz a teremhez.

Viszont az eddigi rendszerben nem volt lehetőség arra, hogy egy egységes felületen áttekinthető legyen a Kabinet aktuális, és jövőben tervezett összes ZH módja. Így természetes volt egy olyan felület létrehozása, amelyhez mind az oktatók, mind az operátorok hozzáférnek, amin látni lehet az összes foglalt vizsga módot. Ez lehetőséget biztosít arra, hogy amennyiben két egymást követő órán is dolgozatot írnak a hallgatók, az oktatók erre fel tudjanak készülni, és az időzítéseknél tudjanak azzal számolni, hogy esetleges csúszások, fennakadások lehetnek a terem kiürítését illetően.

2.2.5. Kommunikáció

Fontos szempont volt, hogy a hirtelen módszerváltozás ne vessen fel kérdéseket, kételyeket sem oktatói, sem operátori szempontból. Ezért implementálásra került egy ímélküldés funkció is. Ez biztosítja, hogy egy, megszokott csatornán keresztül is kommunikáljon a rendszer. Az

új foglalásokról az operátor, illetve a foglaló oktató is levelet kap. Ebben a levélben minden szükséges információ megtalálható ahhoz, hogy az operátor pontosan be tudja állítani a kért foglalást a helyes dátumra és időpontra. Az oktató pedig visszajelzést kap arról, hogy a foglalását elfogadták-e.

Ezzel a kommunikáció mindkét irányban átlátható, tiszta, és nincsenek kérdések azzal kapcsolatban, hogy ténylegesen mi történik a rendszeren belül, működnek-e valóban a funkciók.

2.3. Nem funkcionális követelmények

Egy modern webes alkalmazáshoz már kimondatlanul is támasztanak a felhasználók olyan igényeket, amelyek nem teljesülése esetén rossz felhasználói élményben részesülnek. Annak ellenére, hogy ez az alkalmazás egy jól körülhatárolt felhasználói csoportnak készül, igyekeztünk ezen modern igényeknek is megfelelni.

A NextJS [7] alapértelmezetten, beépítve külön konfigurálás nélkül képes aloldalakat építési időben statikus fájlként létrehozni, így azokon az oldalakon, ahol a tartalom nem változik a betöltési idő konstans időben történik. Emellett optimalizálja a hivatkozások működését [13]. Az oldalak betöltését már akkor elkezdi, mikor a felhasználó még csak a hivatkozás fölé viszi a kurzort, ezzel felgyorsítva a betöltést egy esetleges kattintásnál.

A felhasználói felület tervezésénél szempont volt, hogy ne kelljen sok menüpontot keresgélni vagy több lépésben navigálni, hanem egyszerűen áttekinthető legyen. Viszont fontos volt, hogy az esetleges bővítéseket könnyen lehessen az felhasználói felületen is implementálni.

Annak ellenére, hogy elsősorban nagyobb monitorokra terveztük a használatot, arra is figyeltünk, hogy használható maradjon kisebb, mobiltelefon méretű, kijelzőkön is. Így, bár nem valószínű eshetőség, akár telefonról is lehet vizsgamódot foglalni.

3. TECHNOLÓGIAI ÁTTEKINTÉS

Alább áttekintjük, hogy milyen technológiai alapokon készült el GÉZA. Kitérünk arra, hogy miért pont az adott keretrendszerek vagy könyvtárak lettek választva az adott feladatokra. Emellett átnézzük, hogy milyen külső integrációk találhatók a rendszerben, amelyek nélkül nem lehetne használni, vagy nagyon kellemetlen lenne használni az elkészült eszközt.

3.1. Frontend technológiák

A frontend alapját a NextJS [7] alapjául is szolgáló React [6] adja. A React egy Meta által fejlesztett, elsődlegesen megjelenítési feladatokat ellátó JavaScript könyvtár. A könyvtár az alapvető JavaScript/TypeScript funkcionalitást egészíti ki „HTML-szerű" komponensekkel, amelyeket a fejlesztő készít. Ezen komponensek, amelyek lényegében függvények, pedig tényleges HTML elemeket generálnak, amelyek a böngészőben elhelyezett kód injektál be a DOM fába. A NextJS ezt a funkcionalitást tovább viszi egy kicsit, és vannak olyan részek, amelyeket már alapvetően belegenerál a böngészőnek küldött válaszba, így annak kevesebb dinamikus frissítést, módosítást kell végeznie a tényleges DOM fán.

A stílusos megjelenés alapjául a *TailwindCSS* [14] vizuális keretrendszer szolgál. Ezen keretrendszer filozófiája, hogy nem kész komponensekhez adnak CSS osztályokat, és formázásokat, hanem segédosztályokat ad az egyes formázásokhoz. Erre a keretrendszerre építő *ShadCN* [15] nevű *user interface* (UI) gyűjtemény az, ami a ténylegesen használt komponensekért felelős.

A választás amiatt esett ezekre a technológiákra, mert a NextJS egy kézenfekvő kiinduló keretrendszer volt, amely vonta magával a React-ot is. Ahhoz pedig, hogy ne teljesen a „kályhától" kelljen minden egyes komponenst lefejleszteni kézenfekvő volt, hogy a *ShadCN*-t használni kell, ami pedig hozta magával *TailwindCSS*-t is.

3.2. Backend technológiák

GÉZA háttérét szintén a NextJS [7] alapozza meg a szerver oldali függvényfuttatások segítségével (szakirodalomban: „server actions" [16]). Ezek olyan függvények melyek csak és kizárólag a szerveroldalon léteznek. A felhasználó ezekből a függvényekből csak egy megformált kérést lát a szerver irányába, amit az tud értelmezni, és tudja, hogy melyik függvény kell meghívnia. Így lehetséges, hogy ezek a függvények kódbázis szinten úgy néznek ki, mintha a klienskomponensekkel egy struktúrába lennének rendezve, a valóságban viszont nincs rálátása egyiknek a másikra.

Erre az alapfunkcionalitásra épül rá több felhasznált könyvtár, keretrendszer és külső eszköz, amelyek kulcsszerepet játszanak abban, hogy hibamentesen lehessen a programot használni. Az első ilyen külső keretrendszer a Better-Auth [17], amely a felhasználói bejelentkeztetést végzi. A „LDAP bejelentkezés" fejezetben már említettük, hogy az felhasználók azonosítása az intézeti LDAP [8] szerveren keresztül történik. Ahhoz viszont, hogy az azonosítás folyamata biztonságos legyen a Better-Auth keretrendszert hívtuk

segítésül. A keretrendszer sokkal több funkcionalitásra képes, mint az alkalmazásban szükséges, viszont előnye, hogy egy támogatott és kidolgozott alapot ad. Ez az alap, habár garanciát nem ad arra, hogy hibamentesen fog működni, mégis sokkal megbízhatóbb, mint egy saját fejlesztésű megoldás.

A Better-Auth jár néhány olyan megkötéssel, ami nem ideális a rendszerünk szempontjából, ezekről a „Megkötések függőségek miatt” fejezeben tovább beszélünk. Ezeknek ellenére a használata egyszerű, és a biztonságos felhasználókezelést egyszerűvé teszi, mindemellett bővítésre is lehetőség van benne. Néhány konfigurációval például elérhetővé lehet benne tenni egy úgynevezett „megszemélyesítés” funkciót [18]. Ezzel egy adminisztrátor felhasználó (vagy esetleg egy operátor is) jogosultságot kaphat arra, hogy egy másik felhasználó nevében járjon el. Ez hasznos lehet olyankor, ha az oktató valamilyen hibába ütközik a rendszer használata során, és ebben segítséget kér.

Egy másik könnyen látható előnye a keretrendszer használatának, hogy amennyiben a jövőben bármikor le kell cserélni az LDAP szolgáltatást az Intézetben akkor ahhoz, hogy GÉZA függetlenedjen tőle nem és továbbra is tudjon üzemelni nem kerül majd sok munkába, ugyanis könnyen integrálható egy másik külsős azonosítórendszer, de szükség esetén saját megoldás implementálásra is egyszerű eszközöket nyújt. Így összességében annak ellenére, hogy vannak hátrányai (pl. komplexitás, adatstrukturális megkötések), az előnyei sokkal erősebbek, ami miatt megéri használni.

A háttérrendszerhez kapcsolódóan mindenképp beszélni kell az adattárolásról is. Az alkalmazás az adatokat (felhasználók, foglalások stb.) egy MySQL [19] adatbázisban tárolja. A műveleteket az adatbázison viszont nem direktben végzi, ez ugyanis több biztonsági kockázatot is rejt magában. Az alkalmazást és az adatbázist egy úgynevezett *Object-Relational Mapping* (ORM) rendszer köti össze, a Drizzle [20]. Ez több dolgot is lehetővé tesz: Fejlesztőként ennek a segítségével kódolásnál nem kell ismernünk az adatbázis struktúráját, ugyanis, ha egyszer definiáltuk a sémát, akkor a Drizzle segít az egyes táblák tulajdonságaival, és azok típusaival, másrészről pedig amennyiben valamilyen okból kifolyólag az adatbázis motorját le kellene cserélni (pl. PostgresSQL-re) az alkalmazásban csak a sémadefiníciókat, és a csatlakoztatást kellene módosítani, nem pedig lekérdezéseket keresni a projektben, és átírni másik típusúakra.

3.3. Külső integrációk

Rengeteg külső integrációja van a ZH Adminisztrátornak. Ezekre amiatt van szükség, hogy a lehető legkényelmesebben lehessen használni, ne pedig csak egy további adminisztratív terhet

jelentsen. Ezen külső integrációk – nevük ellenére – legtöbbször valamilyen, az Intézet hálózatán belül elérhető szolgáltatást használnak, vagy építenek arra.

Az egyik legfontosabb ilyen integráció a *Lightweight Directory Access Protocol* (LDAP) [8] integráció. Ez teszi lehetővé, hogy az alkalmazás külön regisztráció nélkül működhessen. Az LDAP egy elterjed, jól támogatott és sztenderdizált protokoll könyvtárszerverek eléréséhez. Ugyanakkor gyakran használják azonosításra, ezesetben is az autentikáció a fő feladata. Az előzőleg bemutatott Better-Auth [17] keretrendszer ugyan beépített („out of the box”) megoldást nem kínál LDAP beléptetésre, viszont egy külsős bővítmény („plugin”) lehetővé teszi az LDAP autentikációt [21].

Az beléptetés menete a megszokott LDAP azonosítás menetét követi [22]. A felhasználó a bejelentkezési felületen megadja a szükséges adatokat. Ezek az adatok biztonságos módon továbbítódnak a szerver felé. A szervernek van hozzáférése a hálózaton elhelyezett LDAP központhoz, amihez megpróbál csatlakozni („bind”) a felhasználó adataival. Amennyiben ez sikerül visszaad az LDAP egy felhasználót a szervernek. Erről néhány fontosabb adatot eltárol GÉZA a saját adatbázisában (pl. ímélcím, felhasználónév, teljes név stb.) amelyek később hasznosak lesznek arra, hogy ne kelljen bizonyos funkciók használatához újra autentikálni az LDAP központon keresztül. Így jelszavakat nem kell tárolnia az alkalmazásnak, csak a jelszavakkal elérhető, amúgy is általában publikus adatokat menti le magának.

A másik ilyen alapköve külső integrációkat tekintve az alkalmazásnak a *Belső Információs Rendszer* (BIR) [9]. Ennek az integrációja amiatt játszik kulcs szerepet, mert enélkül nem lehetne megbízható órarendi adatokat szerezni, anélkül, hogy újra fel kelljen őket vinni. Valószínűleg, és ez csak egy gyenge feltételezés, megoldható lenne, hogy az órarendi adatokat a Neptun rendszeréből [23] szerezze be GÉZA, ennek viszont az lenne a hátránya, hogy az LDAP-ban használt azonosítók nem lennének meg információként. Azaz, valahogyan össze kellene utólag rendelni az órákat az oktatókkal, ami viszont a BIR-ben adatként már szerepel (hiszen annak is az alapja az LDAP). A BIR integráció így kézenfekvő megoldás volt. Technikai szempontból ennek a megvalósítása egészen egyszerű. Az Intézet biztosít az adatbázishoz egy felhasználót, amely egyetlen nézettáblához fér hozzá. Ebben a nézettáblában benne van minden szükséges információ ahhoz, hogy az órarendet GÉZA értelmezni tudja. Minden órához rendelkezésre áll többek között a terem, az oktató vagy oktatók a kezdési időpont és az óra hossza. Ez csak néhány mező az adatok közül, a „Adatbázis felépítése” fejezet bővebben foglalkozik ezzel. Ehhez az adatbázishoz csak olvasási joga van a rendszernek, ez megakadályozza, hogy a teljes BIR-t befolyásoló hibát okozhasson az

adatokban. Illetve, ha valamiért az ottani adatstruktúra le lenne cserélve, akkor csak egy ugyanilyen sémájú táblát kell biztosítani GÉZA részére, és azokkal is tudni fog dolgozni.

Az utolsó nagyobb integráció a levelezés. Ennek az beépítése az eddigiek közül a legegyszerűbb. A szoftver elindításokat környezeti változók segítségével meg kell adni a szükséges *Simple Mail Transfer Protocol* (SMTP) [24] adatokat, ezeket felhasználva pedig GÉZA tud leveleket küldeni. A kiküldött levelek egy sablon alapján készülnek el, melyet a rendszer mindig dinamikusan kitölt az éppen aktuális információkkal. Foglalás esetén például az operátornak és az oktatónak is egy tartalmilag megegyező levél kerül kiküldésre, viszont két külön levélként, ugyanis a jövőben a tartalom eltérő lehet. Ebben a sablonban helye van a foglalt időpontnak (esetleg időpontoknak), a tárgy kódjának, a kért szabálynak és természetesen amennyiben volt feltöltött fájl, amit ki kell majd osztani a számítóképekre, akkor az csatolva lesz a levélhez. Ezen felül az oktató külön értesítő levelet kap arról, ha a kért vizsga módjának státusza megváltozott, azaz elfogadták, vagy elutasították azt.

Nem utolsó sorban pedig az egész rendszer mögött húzódik egy adatbázis is, amely egy újabb külső integráció.

4. ADATBÁZIS FELÉPÍTÉSE

A rendszer alatt egy 6+1 táblából álló adatbázis szolgáltatja az adatok tárolását, és rendelkezésre állását. Az jelenleg használat adatbázis motor pedig MySQL [19]. Ez egy kényelmes választás volt, az Intézeti szervereken több adatbázis is fut MySQL motort használva, szóval így nem szükséges újabb szoftverek telepítése.

A 6+1 táblából az „plusz 1” rész egy nézettábla a BIR [9] adatbázisán. Ennek sémáját egy a „Adatbázis részletes elemzése” fejezetben tárgyaljuk. Ebből a táblából csak adatolvasás történik.

A maradék 6 tábla az, amelyet az alkalmazás hoz létre, és használ is adattárolás céljából. Ezek pedig a következők: *user, session, account, verification, rule, booking*. Az első 4 tábla ezek közül a Better-Auth működéséhez szükséges adattáblák [25]. Ezen táblák mezőire minimális a befolyása az alkalmazásnak. A maradék 2 tábla pedig az, ami ténylegesen a működést kiszolgálja.

A *user* tábla egy felhasználó adatait tárolja. Ez az egyetlen tábla, amelyet közvetlenül lehet a Better-Auth keretrendszeren befolyásolni. A következő mezőket tartalmazza: *id, name, email, email_verified, image, role, banned, ban_reason, ban_expires, gid_number, fullname, display_name, created_at, updated_at*. Ezek közül az *id, name, email, email_verified, image, created_at, updated_at* mezők a Better-Auth alapvető architektúrájából jönnek [25], a *role,*

banned, ban_reason, ban_expires pedig a jogosultságkezelő bővítmény hozománya [18], végül pedig a mezők *gid_number, fullname, display_name* az LDAP-ból kerülnek átmentésre bejelentkezéskor.

A *session* tábla egy munkamenet adatait tárolja. Ezen mezőket mind a használt keretrendszer hozza létre, és kezeli: *id, expires_at, token, created_at, updated_at, ip_address, user_agent, user_id, impersonated_by*. A legutolsó mezőt a jogosultságkezelő bővítmény adja hozzá a táblához [18].

A *rule* tábla tárolja az alkalmazható szabályokat az egyes foglalások esetén. Ezen tábla attribútumai: *id, name, description, action, created_at, updated_at*.

Végül pedig a *booking* tábla mezői: *id, user_id, rule_id, classroom, course, start_time, end_time, status, custom_request, created_at, updated_at*. Ebben a táblában az rekordok foglalásokat jelentenek.

Az *account* és *verification* táblákról további leírás a „Megkötések függőségek miatt” fejezetben találhatók.

4.1. Adatbázis részletes elemzése

Annak ellenére, hogy a fent leírt adattáblák nagyrészét a külső keretrendszer kezeli, érdemes ismertetni a működését, ugyanis az alkalmazás szempontjából lényeges, és a későbbiekben a dolgozatban is hivatkozott adatokat tárolnak.

A *user* értelemszerűen tábla egy felhasználó összes tulajdonságát raktározza. Ezt a táblát, habár a keretrendszer hozza létre, lehet szerkeszteni, azaz lehet hozzáadni plusz attribútumokat (elvételre nincs lehetőség) [25]. A következő tulajdonságokat a keretrendszer magától létrehozza:

- *id*: szöveges, generált azonosító, elsődleges kulcs,
- *name*: szöveges, LDAP felhasználói név, egyedi,
- *email*: szöveges, ímélcím az LDAP-ból átvéve, egyedi,
- *email_verified*: igaz/hamis érték, az ímélcímet megerősítette-e a felhasználó, az alkalmazás nem használja (feltételezi, hogy az ímélcím az LDAP-ban valós),
- *image*: szöveges, profilkép, az alkalmazás nem használja,
- *created_at*: dátum és időpont, azt tárolja, hogy a rekord mikor jött létre,
- *updated_at*: dátum és időpont, azt tárolja, hogy a rekord mikor frissült utoljára.

Ezen felül a Better-Auth-hoz használt egyik kiegészítő [18], amely a jogosultságkezelést végzi hozzáad még néhány mező, amelyek a következők:

- *role*: szöveges, a felhasználó szerepköre a rendszerben, lehetséges érkékek: „*admin*”, „*operator*”, „*teacher*”,
- *banned*: igaz/hamis érték, a felhasználó ki van-e tiltva a rendszerből, a rendszerből tervezetten senkit nem kell kitiltani, szóval nem használt mező,
- *ban_reason*: szöveges, a kitiltás oka, a rendszerből tervezetten senkit nem kell kitiltani, szóval nem használt mező,
- *ban_expires*: dátum és időpont vagy null, meddig érvényes a kitiltás, a rendszerből tervezetten senkit nem kell kitiltani, szóval nem használt mező.

Az alkalmazás bejelentkezéskor átvesz néhány adatot az LDAP [8] rendszeréből, amit szintén a felhasználói táblába ment el, ezek a felsoroltak:

- *gid_number*: szám, felhasználói csoportazonosító az LDAP-on belül,
- *fullname*: szöveges, teljes, LDAP-ban tárolt név,
- *display_name*: szöveges, az LDAP-ban beállított megjelenítési név.

A *session* tábla egy bejelentkezés utáni munkamenet adatait tárolja. Ezen mezőket mind a használt keretrendszer hozza létre, és kezeli is, az attribútumok a következők:

- *id*: szöveges, generált azonosító, elsődleges kulcs,
- *expires_at*: dátum és időpont, amikortól a munkamenet már nem érvényes,
- *token*: szöveges, generált szöveg, amely információkat hordoz a felhasználóról, és a munkamenetéről, egyedi,
- *created_at*: dátum és időpont, azt tárolja, hogy a rekord mikor jött létre,
- *updated_at*: dátum és időpont, azt tárolja, hogy a rekord mikor frissült utoljára,
- *ip_address*: szöveges, a munkamenet elindításakor milyen IP címe volt a felhasználónak,
- *user_agent*: szöveges, a munkamenetet milyen eszközről indították (az eszköz saját maga által meghatározott fejlécének tartalma),
- *user_id*: szöveges, külső kulcs, a felhasználó rekordjára mutat,
- *impersonated_by*: szöveges, annak a felhasználónak az azonosítóját tárolja aki ezt a munkamenetet úgy hozta létre, hogy saját fiókjából (pl. adminisztrátor) megszemélyesíti éppen a tényleges tulajdonosát a munkamenetnek.

A *rule* tábla tartalmazza az operátorok által már létrehozott szabályokat, amelyeket a felhasználó majd a foglaláskor ki tud választani. A következő mezőkkel rendelkezik:

- *id*: szám, a szabály egyedi azonosítója, automatikusan növekvő, elsődleges kulcs,
- *name*: szöveges, a szabály néhány szavas megnevezése,

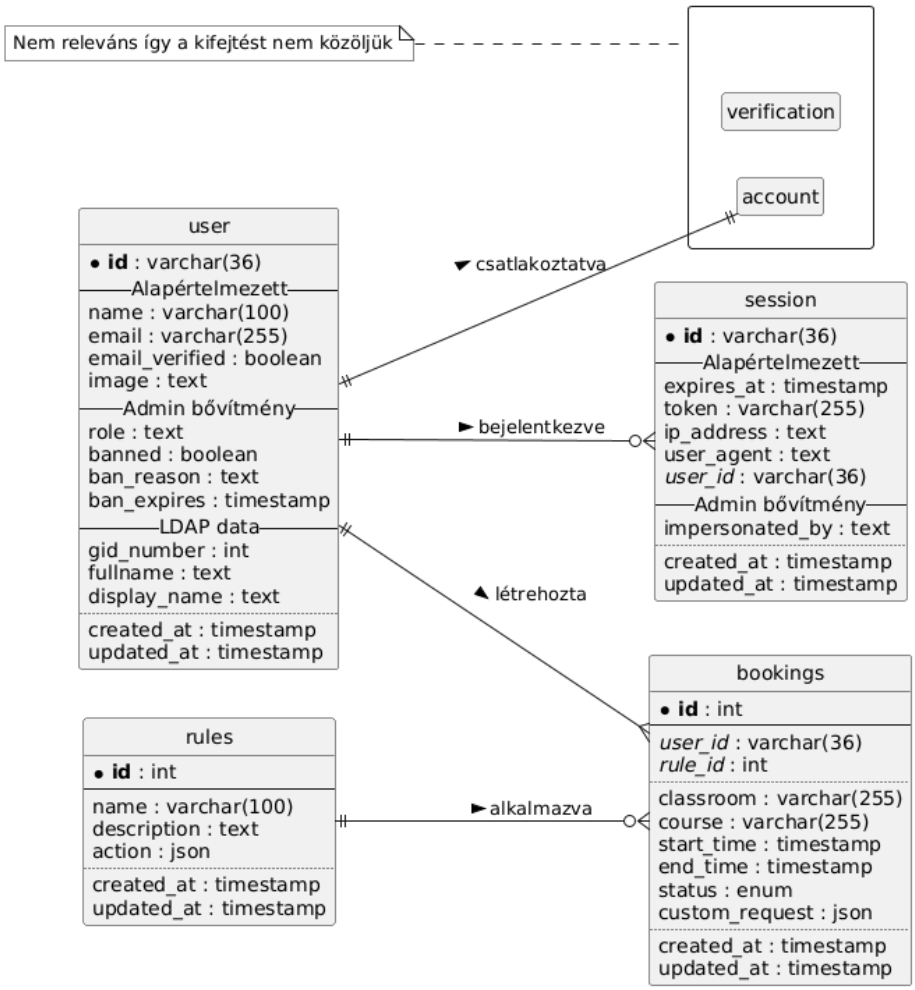
- *description*: szöveges, a szabály hosszabb leírása, pontosan mit csinál, mit tilt és mit engedélyez,
- *action*: szöveges (JSON), egy objektum, amely a jövőbeli fejlesztéseknek ad teret, jelenleg tárolja a parancsfájl nevét, amely a szabályhoz tartozik,
- *created_at*: dátum és időpont, azt tárolja, hogy a rekord mikor jött létre,
- *updated_at*: dátum és időpont, azt tárolja, hogy a rekord mikor frissült utoljára.

Az utolsó tábla, amely jelen kontextusban számottevő a *booking* felépítése a következő:

- *id*: szám, a foglalás egyedi azonosítója, automatikusan növekvő, elsődleges kulcs,
- *user_id*: szöveges, külső kulcs, azt tárolja, hogy melyik felhasználó hozta létre a foglalást,
- *rule_id*: szám, külső kulcs, azt tárolja, hogy a foglaláshoz melyik szabály tartozik, vagy null egyedi kérés esetén,
- *classroom*: szöveges, a BIR-ből kinyert adatok alapján melyik gépteremhez tartozik a foglalás,
- *course*: szöveges, a foglaláshoz tartozó kurzus Neptun kódját tárolja,
- *start_time*: dátum és időpont, az a pillanat amikor el kell indítani az adott szabályt,
- *end_time*: dátum és időpont, az a pillanat amikor le kell zárni az adott beállítást,
- *status*: háromértékű mező, azt tárolja, hogy milyen állapotban van a foglalás, értékei lehetnek: „*pending*”, „*permitted*”, „*declined*”,
- *custom_request*: szöveges (JSON), amennyiben nem egy előre definiált szabályt választott az oktató, hanem egyéni kérése volt, egy objektumot tárol, amely a legfontosabb beállításokat tárolja,
- *created_at*: dátum és időpont, azt tárolja, hogy a rekord mikor jött létre,
- *updated_at*: dátum és időpont, azt tárolja, hogy a rekord mikor frissült utoljára.

A leírás jobb áttekinthetőségének érdekében a 4.1. ábra mutatjuk a felépítését az adatbázisnak, kapcsolatok jelölésével együtt. Az ábra nem egy sztenderd jelölésrendszert használ, elsősorban az az intuitív megértésre épül. A bal felső sarkában egy megjegyzés található, ez nem része az adatbázisnak. Az entitások közötti vonalak a következőképpen értendőek: dupla függőleges vonal jelöli a kapcsolat „egy” részét, a háromfejű vonalvég jelöli a kapcsolat „N” részét, amennyiben a háromfej előtt kör található a kapcsolat azon része megengedi „nulla darab” egyedet is (azaz nem létező kapcsolatot). Az attribútumok elválasztásánál a következő szempontokat vettük figyelembe: az elsődleges kulcs mindig külön szerepel, a megjegyzés az elválasztási vonalban azt jelöli, hogy a következő

tulajdonságok milyen forrásból jönnek, ahol van ott a létrehozást, és az utolsó frissítést tároló mezők az átláthatóság kedvéért az egyed végére kerültek, pontozott elválasztással. Dőlt betűvel pedig jelöltük a külső kulcsokat.



4.1. ábra [26]

A *Belső Információs Rendszer* adatbázisából a következő adatséma az, amelyhez GÉZA hozzáférést kapott:

- *user_id*: szám, egyedi azonosítója a felhasználónak a BIR-ben,
- *user_email*: szöveges, az LDAP azonosító,
- *user_name*: szöveges, a felhasználó neve az LDAP-ból (feltételezhetően),
- *teacher_id*: szám, tanárazonosító,
- *teacher_department_unique_id*: szám, tanszékazonosító,
- *teacher_active*: igaz/hamis érték, a tanár tanít-e,
- *course_id*: szám, a kurzus egyedi azonosítója,
- *course_neptun_id*: szöveges, a kurzus Neptun azonosítója (szintén egyedi),
- *course_unique_name*: szöveges, a kurzus egyedi neve,

- *course_full_name*: szöveges, a kurzus megjelenített neve,
- *course_type*: szöveges, a kurzus típusa (pl. előadás, gyakorlat)
- *course_scool*: szöveges/karakter, nappali vagy levelező kurzus-e,
- *course_hour*: szám, a kurzus hossza (órákban mérve),
- *course_prefill*: szám, mennyi idő szükséges a kurzus előtt szabadon (feltételezhetően),
- *course_department_unique_id*: szám, a kurzust tartó tanszék azonosítója,
- *course_division_dephelper*: szám, a jövőbeli fejlesztések okán van benne a sémában,
- *week_id*: szám, a tanhét azonosítója,
- *week_week*: szöveg, az azonosító, és a dátumokból összeállt szöveges megjelenített mező,
- *week_active*: igaz/hamis érték, szünet-e a hét,
- *week_date*: dátum, a hét kezdődátuma (hétfő),
- *course_starttime_starthour*: szám, 0-23 közötti szám, azt mondja meg, hogy mikor kezdődik a kurzus,
- *course_starttime_day_id*: szám, azt mondja meg, hogy a héten belül melyik napon van a kurzus,
- *classroom_neptun_id*: szöveges, a tanterem Neptun-beli azonosítója,
- *classroom_unique_name*: szöveges, a tanterem egyedi neve,
- *classroom_full_name*: szöveges, a tanterem megjelenítendő neve.

Ezekben a megkapott adatokban vannak olyan mezők, amelyeket jelenleg GÉZA nem használ, viszont egy esetleges jövőbeli fejlesztés esetén hasznosak lehetnek, ezért, hogy akkor ne kelljen a struktúrán módosítani már előre hozzáférést kapott hozzá. Az is látható, hogy vannak benne redundáns mezők is, ezek oka az, hogy lehetővé tegyünk minél több módosítási lehetőséget.

4.2. Normalizálási elemzés

A Better-Auth [17] által kezelt, illetve BIR-ből [9] jövő adatok normalizálási elemzésétől eltekintünk, ugyanis azokra ráhatásunk minimális, vagy egyáltalán nincs is. A következő fejezetben megemlítésre kerül, hogy vannak bennük redundáns adatok, és fölösleges táblák, illetve mezők.

A GÉZA által létrehozott két táblának pedig a sémáját a következő módon lehet felírni (aláhúzással jelölve az elsődleges kulcs, dőlt betűvel szedve pedig a külső kulcsok):

Rule(id, name, description, action, created_at, updated_at)

Booking(id, *user_id*, *rule_id*, classroom, course, start_time, end_time, status, custom_request, created_at, updated_at)

Az 1. normálforma [27] követelményei, miszerint ne minden attribútum atomi teljesül, mindkét sémában egyszerű értékeket reprezentálnak a tulajdonságok.

A 2. és 3. normálforma [27] vizsgáláshoz már a függőségeket is elemezni kell, viszont ez egyszerűen megtehető. A szabályok esetében nem feltétlenül szerencsés azt mondani, hogy a neveknek egyedinek kell lennie, hiszen elképzelhető szabály, aminek ugyanaz a neve, viszont leírásban eltérnek. Az mesterséges egyedi azonosítótól eltekintve bármely valódi részhalmazát is vesszük az attribútumoknak nem találunk újabb lehetséges kulcsot, így szükséges az mesterséges azonosító generálása. Foglalások estén pedig látható, hogy habár a kurzus terme, és kurzuskód összefügg viszont nem minden esetben következik egyik a másikból (teremváltozás esetén például sérül a szabály). A szabály azonosítója talán összefügg a kurzus azonosítójával, viszont itt is lehetnek kivételek. Hasonlóan, eltekintve a mesterséges egyedi azonosítótól nem találunk olyan valódi részhalmazt a foglalás attribútumai között, amely meghatározná a teljes rekordot. Tranzitív függőséget pedig egyik sémában sem találunk, minden közvetlenül a kulcstól függ.

4.3. Megkötések függőségek miatt

Az előző fejezetben nem történt elemzése a Better-Auth [17] által létrehozott tábláknak. Ennek az az oka, hogy a sémái már 3. normálformát nem érik el, emiatt optimalizálásra szorulnának. Erre viszont nem ad lehetőséget a keretrendszer.

A felhasználó adatait tároló táblán könnyen látható, hogy tranzitív függés van benne (nem is egy). A mesterséges azonosító mellett a név és az ímélcím is egy jó kulcsa lenne a táblának. Így a mesterséges azonosítóra nem lenne semmi szükség, lehetne a név is elsődleges kulcs (ugyanis ezt az LDAP [8] garantálja, hogy nincs két egyforma azonosító).

Viszont ezt az apró redundanciát megengedve, inkább amellett döntöttünk, hogy egy megbízható, jól dokumentált és nagy támogatással bíró keretrendszer kezelje az azonosítási feladatokat. Ennek hosszútávon több előnye van (mint előzőleg már tárgyaltuk), mint az a minimális tárhelyveszteség, amit okoz.

Az eddigiekben nem esett szó, az *account* és *verification* táblákról. Ezek is a „szükséges rossz” kategóriába tartoznak. Mindkettő elengedhetetlen ahhoz, hogy a Better-Auth

használható legyen, viszont valós felhasználása nincs az alkalmazásban. Az *account* tábla tárolná a felhasználóknak a különböző fiókjait, ha lenne lehetőség például Google vagy egyéb fiókokkal belépni. Mivel csak és kizárólag LDAP azonosítás lehetséges az alkalmazásban, minden felhasználónak egyetlen fiókja lesz ebben a táblában. A *verification* táblát pedig arra használná a keretrendszer, hogy ha szükséges lenne az ímélcímeket megerősíteni, akkor ebben a táblában tárolná a generált azonosítókat, amelyeket elküldene a megadott ímélcímekre. Viszont ez a funkcionalitás ki van kapcsolva, így ez a tábla végig üres marad.

5. RENDSZERARCHITEKTÚRA

GÉZA architekturális felépítése szorosan követi a már bemutatott technológiai döntéseket: a rendszer alapját a NextJS [7] keretrendszer adja, amely egyszerre biztosítja a felhasználói felület megjelenítését és a háttérben futó üzleti logika kiszolgálását. A keretrendszer által használt újabb, úgynevezett „App Router" [28, 29] architektúra lehetővé teszi, hogy a fájlrendszer felépítése egyben a webes útvonalakat is definiálja, így az alkalmazás áttekinthető és jól követhető struktúrát kap.

A projektben az alkalmazás forráskódja az „src" könyvtár alatt található, ezen belül a „src/app" mappa tartalmazza az egyes oldalaknak megfelelő útvonalakat és elrendezéseket, míg a „src/components" könyvtár az újrafelhasználható megjelenítési elemeket fogja össze. A „src/lib" könyvtárban kaptak helyet az adatbázis-hozzáférést, autentikációt és egyéb üzleti logikát megvalósító függvények, az „src/hooks" pedig az egyedi React hook-okat tárolja, amelyek elsősorban kliensoldali állapotkezelési feladatokat látnak el. A NextJS által nyújtott struktúra így egy természetes rétegezést eredményez: az útvonalak, a megjelenítés és az üzleti logika fizikailag is elkülönülnek egymástól.

Az „App Router" egyik fontos eszköze az úgynevezett „route group" [30] (oldalcsoport), amelyet zárójelbe tett mappanév jelöl (például „(user)"). Ezek a mappák segítenek az útvonalak logikai csoportosításában anélkül, hogy a tényleges URL-ben megjelennének, így GÉZA esetében az „src/app/dashboard/(user)/…" útvonalak például felhasználói (tanári) nézeteket foglalnak össze (órarend, foglalások, áttekintések), miközben az URL-ben továbbra is csak a „/dashboard/…" struktúra látszik. Ennek köszönhetően a rendszerben egyszerre lehet külön kezelni az oktatói, operátori és adminisztrátori felületeket.

5.1. Szerver- és kliensoldali komponensek

A NextJS jelen verziója már alapértelmezetten úgynevezett „szerveroldali komponenseket" használ [28], ami azt jelenti, hogy a legtöbb oldal és elrendezés a szerveren kerül

összeállításra, és a böngésző egy már a kész oldalt kapja meg, amit csak meg kell jelenítenie. Kliensoldali komponens akkor jön létre, ha a fájl tetején megjelenik a „"use client"” direktíva; ilyen esetben a komponens a böngészőben fut, teljes hozzáféréssel a React állapotkezeléshez, eseménykezelőkhöz és a böngésző API-khoz. A két komponensfajta együttműködése teszi lehetővé, hogy GÉZA egyrészt gyorsan betöltődjön, másrészt interaktív, kényelmesen használható felhasználói felületet nyújtson. [31]

A szerveroldali komponensek feladata GÉZA-ban elsősorban az, hogy az egyes oldalak betöltésekor „összegyűjtsék” az összes szükséges információt: ellenőrizzék, hogy a felhasználó be van-e jelentkezve, és előkészítsék azokat az adatstruktúrákat, amelyekkel majd a kliensoldali komponensek dolgoznak. Jó példa erre a tanári felületen látható órarend oldal („src/app/dashboard/(user)/schedule/page.tsx”, az 5.1. ábra látható), amely először az azonosításhoz használt Better-Auth keretrendszeren keresztül lekéri az aktuális felhasználó munkamenetét, majd egyszerre kéri le a BIR-ből származó órarendet és a Kabinet által használt szabályokat, végül pedig ezeket adja át a megjelenítésért felelős komponensnek. Így az oldal a felhasználó szempontjából már egy kész, koherens képet mutat, amikor betölt.

```tsx
export default async function Page() {
  let session;

  try {
    session = await auth.api.getSession({
      headers: await headers(),
    });
  } finally {
    if (!session) {
      redirect("/login");
     }
  }

  const [userSchedule, rules] = await Promise.all([
    getUserSchedule(session.user.name),
    getRules(),
  ]);

  return (
    <SchedulePage
      classes={userSchedule}
      rules={rules}
      userLdap={session.user.name}
    />
  );
}
```

5.1. ábra

A kliensoldali komponensek GÉZA-ban ott jelennek meg, ahol valódi interaktivitásra van szükség. A tényleges megjelenített órarend oldal például „"use client"” direktívával indul („src/app/dashboard/(user)/schedule/schedule-page.tsx”, az 5.2. ábra látható egy részlete), mivel egy interaktív naptárat jelenít meg, amelyben a felhasználó eseményekre kattinthat, és

ezek alapján foglalási párbeszédablakokat tud megnyitni. Ebben a komponensben React állapotkezelés („useState”, az ábrán 6. sor) és életciklus-kezelés („useEffect”, az ábrán 10.-12. sorok) segítségével alakul át a szerverről érkező órarendi lista egy olyan eseménylistává, amelyet a naptárkomponens már meg tud jeleníteni, illetve kezeli azt is, hogy mikor nyíljon meg vagy záródjon be a foglalási felület. Ugyancsak kliensoldali komponens az „app-sidebar.tsx”, amely a bejelentkezett felhasználó szerepköre (oktató, operátor, admin) alapján dinamikusan állítja össze a menüpontokat.

```tsx
"use client";

//import ...

export function SchedulePage(/* props */) {
  const [isBookingDialogOpen, setIsBookingDialogOpen] = useState(false);
  const [bookingEvent, setBookingEvent] = useState<CalendarEvent | undefined>(undefined);
  const [events, setEvents] = useState<CalendarEvent[]>([]);

  useEffect(() => {
    setEvents(classToEventTransformer(classes));
  }, [classes]);

  return /* ... */;
}
```

5.2. ábra

A két komponensfajta közötti együttműködés jól látható a „src/app/dashboard/layout.tsx” felépítésén. Ez az elrendezés szerveroldali komponensként ellenőrzi, hogy létezik-e érvényes munkamenet, és ha nincs, átirányítja a felhasználót a bejelentkezési felületre. Amennyiben viszont van érvényes munkamenet, a komponens egy kliensoldali *SidebarProvider* és *AppSidebar* köré „csomagolja” a tartalmat. Ez az elrendezés, a gyökér „layout.tsx” fájllal együtt (5.3. ábra), gondoskodik arról, hogy az olyan, interaktivitást igénylő elemek, mint a téma (világos/sötét mód) [32] vagy az értesítések (Sonner) [33], minden oldalon egységesen, ugyanúgy működjenek, miközben az adatok lekérdezése továbbra is a szerveren történik. Így GÉZA architektúrája egyszerre marad biztonságos, gyors és a felhasználók számára kényelmesen használható.

```tsx
export default function RootLayout(/* props */) {
  return (
    <html lang="hu" suppressHydrationWarning>
      <body
        className={`${geistSans.variable} ${geistMono.variable} antialiased`}
      >
        <ThemeProvider attribute="class" defaultTheme="system" enableSystem>
          {children}
          {/* Ide kerül majd a következő layout is, amelyben a SidebarProvider van */}
          <Toaster position="bottom-right" expand richColors closeButton />
        </ThemeProvider>
      </body>
    </html>
  );
}
```

5.3. ábra

5.2. Jogosultságkezelés

A jogosultságkezelés kulcskérdés egy ilyen rendszerben. Jogosulatlan hozzáférés bizonyos felületekhez, információkhoz hosszú távon okozhat nagy problémákat. Egyetlen oktató sem szeretné, ha az egyik hallgatója például lemondaná a kért vizsga módot. Jelen rendszerben talán nem is lenne ez akkora probléma, ugyanis a tényleges beállítása a kéréseknek manuális, viszont, ha egy továbbfejlesztett, félig vagy teljesen automatikus rendszerben történik ilyen, akkor az nem egy szerencsés eset.

Ezért GÉZA használ a Better-Auth-hoz egy bővítményt [17, 18], amiről már egy párszor említés szinten beszéltünk, amely a jogosultságokat képes kezelni. Ez a bővítmény alapvetően két szerepet („role”) definiál, egy felhasználóit, és egy adminisztrátorit. Ezen szerepek viszont tetszés szerint bővíthetők, felülírhatók, és jogköröket is rendelhetünk hozzájuk. Ezen jogkörök felépítése úgy néz ki, hogy van egy úgynevezett erőforrás („resource”), amelyhez rendelhetünk különböző jogokat („permission”). Nincs semmilyen megkötés arra vonatkozóan, hogy hány erőforrásunk lehet, illetve azoknak hány jogosultsággal kell rendelkeznie. Szóval hosszútávon egy jól bővíthető rendszert kapunk.

Jelenleg a következő erőforrások, és hozzájuk tartozó jogosultságok léteznek, ahogy az 5.4. ábra is mutatja:

- user: create, list, set-role, ban, impersonate, delete, set-password,
- session: list, revoke, delete,
- booking: make, decide, view,
- schedule: viewSubject,
- rules: read, create, update, delete.

Ezek közül az első kettő beépített, a maradék 3 pedig saját kezűleg hozzáadott erőforrás.

```
const statement = {
  ...defaultStatements,
  booking: ["make", "decide", "view"],
  schedule: ["viewSubject"],
  rules: ["read", "create", "update", "delete"],
} as const;
```

5.4. ábra

Megjegyzés: A „schedule” erőforrást, és a hozzá tartozó jogköröket a rendszer még sehol sem használja, ez csak egy jövőbeli fejlesztést készít elő.

5.3. Funkciók a rendszerben

Ahogy korábban említettük GÉZA egy alapot ad egy későbbi teljesen automatikus, emberek által csak felügyelt zárhelyi dolgozatokhoz használt vizsga mód ütemezését és menedzselését ellátó rendszerhez. Ennek az első lépését valósítja meg a rendszer, melyhez néhány alapfunkció lett implementálva.

Az adatokat átveszi a BIR [9] rendszerből, minden oktató láthatja a saját órarendjét, és azokhoz tetszőlegesen foglalhat ZH módot. Kényelmi funkcióként belekerült, hogy egy kurzushoz, egy foglaláson belül a félév valamennyi időpontjára lehet foglalni, ugyanazokkal a beállításokkal. Ez hasznos lehet, ha valaki előre tervezetten tudja az összes dolgozat időpontját.

A foglalásokról a rendszerüzemeltetők levelet kapnak, mely alapján be tudják állítani a kért üzemmódot. Ebben a levélben benne van minden szükséges adat, ami kell ahhoz, hogy a fölösleges ímélezést el lehessen kerülni. Nem kell visszakérdezni időpontokra, szabályokra, hanem minden egyszerre érkezik.

Ezeket a foglalásokat jóváhagyhatják beállítás után, ezzel jelezve, hogy az oktató számíthat arra, hogy működni fog az óráján a kért rendszer.

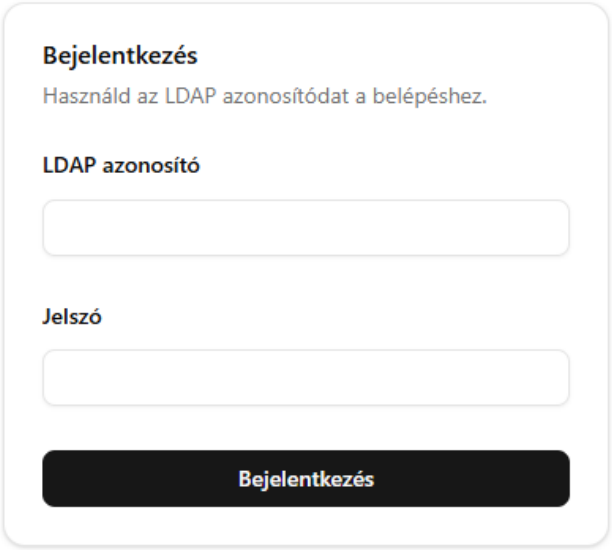
A szabályokat előre fel kell vinniük az operátoroknak, egy néhány soros leírással, az oktatók ezek közül tudnak választani foglaláskor. Amennyiben nem találnak megfelelőt, igényelhetnek saját elképzelésük szerint egyet, amelynél szabadszavas leírást adhatnak meg, viszont néhány alap kérdésre is válaszolniuk kell.

A teljes Kabinet termeire nézve látható egy naptár, amiben az összes foglalás, és azok státusza követhető nyomon. Erre minden bejelentkezett felhasználó rálát.

6. FELHASZNÁLÓI ÚTMUTATÓ

Két fontos felhasználó csoport van, akik számára fontos útmutatást adni, az egyik ilyen csoport a tanárok, a másik pedig az operátorok. A rendszerben harmadik felhasználócsoportként az adminisztrátorok is jelen vannak. Számukra elérhetőek az operátori és tanári funkciók egyszerre.

Minden esetben bejelentkezéssel kell kezdeni. Ehhez értelemszerűen ki kell tölteni a bejelentkező felületen az adatokat (6.1. ábra). Az LDAP azonosító az intézeti azonosító helye, a jelszó pedig az ahhoz tartozó jelszó. Ilyen azonosítója az operátoroknak és az oktatóknak is egyaránt van.



6.1. ábra

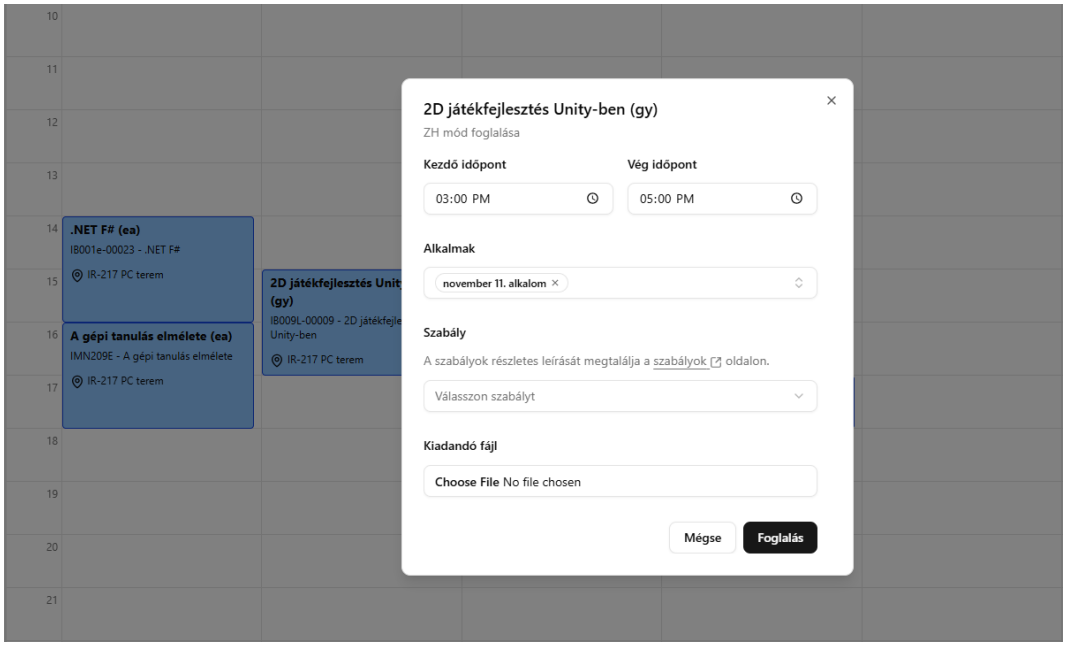
Mielőtt a tényleges útmutatókra térünk meg kell jegyezni a következőt: Az adatok, amelyek a képernyőképeken a valóság alapján készültek, viszont fiktívek.

6.1. Tanári útmutató

Oktatóként az órarendjét tartalmazó oldal fogadja a felhasználót, amelyben naptárszerű navigáció van megvalósítva [34]. Ezen az oldalon láthatja azokat a kurzusokat, amelyekhez hozzá van rendelve, és ezen az oldalon foglalhat magának új vizsga módot is egy kurzusára kattintva (6.2. ábra).

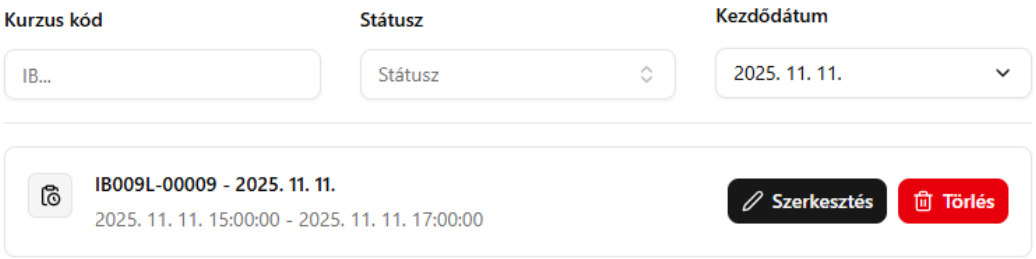
A foglaló felület egy olyan űrlapot tartalmaz, amely mind kliens mind szerveroldalon ellenőrizve van, így hibás adatokat még véletlenül sem tud megadni a felhasználó. Az űrlapon először meg kell adnia az órán belül kezdő- és végidőpontot, utána (ha szeretné) kiválaszthat több alkalmat is egyszerre, amelyekre foglalni szeretne. A következő elemben ki kell választania az alkalmazni kívánt szabályt, vagy meg kell adnia, hogy egyéni szabályt szeretne

(ebben az esetben megváltozik az űrlap kinézete, és több adat megadására szólítja fel a felhasználót). Utolsó elemként pedig lehetősége van fájlt felölteni, amelyet ki szeretne adni a hallgatók gépeire. Ez természetesen opcionális.



6.2. ábra

A kérés leadása után a „Kért ZH módok” menüpontban megjelenik a kérés, amelyet leadott a felhasználó. Van egy szűrő az oldal tetején, amely segíti a navigációt (6.3. ábra). Amíg a kérés nem kerül elfogadásra a felületen lehetőség van időpontok módosítására (itt is csak a kurzus kezdő- és végidőpontja közt lehet értékeket megadni), egyéb szerkesztési lehetőség nincs.

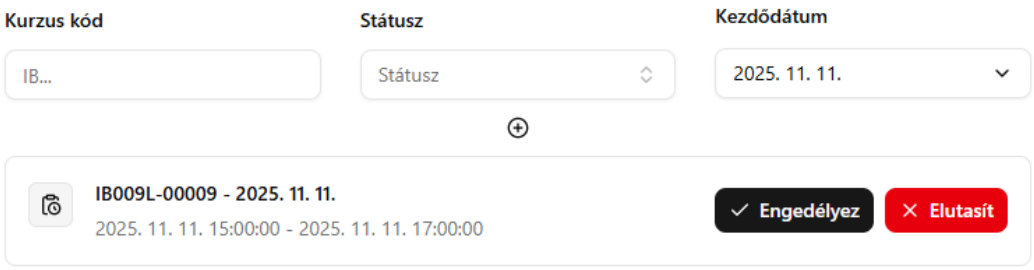


6.3. ábra

A leadott igény ekkor már ímélben az operátornak elküldésre került, aki megkezdheti a jóváhagyás folyamatát.

6.2. Operátori útmutató

Operátorként bejelentkezés után egyből a kéréseket megjelenítő oldalon találjuk magunkat. Ez a felület nagyon hasonlít a tanári felülethez, viszont van néhány fontos eltérés (6.4. ábra).



6.4. ábra

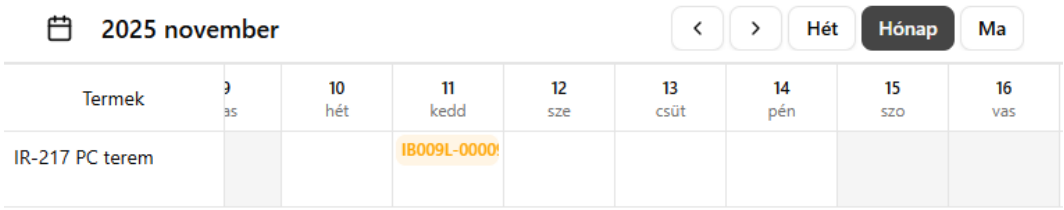
Látható, hogy itt a gombok engedélyezni, illetve elutasítani tudják a kéréseket. Miután az operátor beállította a kért módot, rányomhat az „Engedélyez” gombra, amellyel visszajelzést küld az oktató számára, hogy megtörtént a beállítás, és számíthat arra, hogy elindul majd az üzemmód. Elutasítás esetén pedig szintén tájékoztatva lesz az oktató arról, hogy valami nem volt rendben a foglalásával.

Ami még különbség ezen a felületen, az az, hogy egy „plusz” ikon is található a szűrők alatt. Ezzel lehet egyéni, teljesen szabad foglalást felvenni, amennyiben szükséges ez valamilyen okból kifolyólag, ennek a felülte hasonló a 6.2. ábra láthatóhoz.

Ezen felül egy külön menüpontban az operátoroknak lehetősége van felvenni új szabályokat, és ahhoz leírásokat. Ezek a felvett szabályok és leírások az oktatóknak a foglaláskor lesznek megmutatva, ahhoz, hogy ki tudják választani a számukra megfelelőt.

6.3. Kabineti áttekintés

Mindkét felhasználói csoportnak van hozzáférése egy „Kabineti áttekintés” nevű menüponthoz, ahol egy naptárnézetet [34] látnak (6.5. ábra). Itt géptermekre bontva látható, hogy mely napokon és a napokon belül mely időszakokban van egy teremben futó ZH mód.



6.5. ábra

Ezen a felületen színekkel (piros, zöld, sárga) jelölve van a három lehetséges státusza az egyes foglalásoknak (elutasítva, elfogadva, függőben).

7. ÖSSZEGZÉS

A következőkben áttekintjük hogyan lehet integrálni ezt a rendszert a már meglévőbe. Ezen felül megnézzük, hogy milyen lehetőségek vannak a rendszer jövőjét tekintve, és összesítjük mit is valósít meg GÉZA.

7.1. Integráció

Az integráció kérdése a Kabinetben jelenleg futó rendszerhez jelenleg nem számottevő kérdés. Ugyanis üzemeltetési szemmel egy weboldalt kell csak kiszolgálni, amihez kell egy adatbázist kapcsolni, hozzáférést biztosítani a BIR adatbázisához, és megadni néhány konfigurációt az ímélezéhez.

Erre több megoldás is elképzelhető: egy egyszerű webszerver és MySQL adatbázis, amelyhez a szükséges adatok környezeti változókban vannak eltárolva, vagy egy virtualizált Docker [35] környezet, mely szintén környezeti változókként kapja meg a szükséges adatokat.

Az integráció kérdése akkor válik majd igazán fontossá, mikor a rendszer teljesen magától fog majd beállításokat végezni, ezekre viszont majd akkor keresünk választ.

7.2. Potenciál a jövőre nézve

Már a mostani rendszer is úgy épült fel, hogy szem előtt volt tartva az, hogy a jövőben egy nagyobb kaliberű eszköz lesz belőle. Emiatt a fejlesztési lehetőségek nincsenek lezárva a rendszerben.

Ami lehet az modulárisan lett felépítve, a menürendszertől elkezdve, ami dinamikus épül fel, és hozzáadni egy új menüpontot annyiból áll, hogy egy tömbbe felvesz a fejlesztő egy új elemet, egészen a foglalás leadásáig, ahol kicserélni a levélküldést egy komolyabb funkcionalitást ellátó függvényre csak néhány sor átírását jelenti.

7.2.1. Ideális vízió

A mai világban véleményem szerint nincs teljesen kész szoftver. Úgy gondolom, hogy GÉZA sem lesz sosem kész. Viszont létezik olyan állapot, amely ideális, és minden igénynek megfelel.

GÉZA esetében ez az állapot a következő funkciókat is tartalmazza: teljesen automatizált foglalás, akár egyedileg kért, előtte még sosem látott szabályokra, kérések a termek bizonyos számítógépeire, anélkül, hogy az összes számítógép szabályok alá lenne vonva, tanári számítógépről, akár a teremből a dolgozat közben ellenőrizhető beállítások, amely magában foglalja azt is, hogy szükség esetén a tanár extra időt adhat a diákoknak, automatizmus amely a jelenleg használatban lévő, és esetlegesen később létrehozott számonkérő rendszerekhez csatlakozik (Bíró, Bíró3), és lehetővé teszi, hogy a kéréseket már azokban a rendszerekben el lehessen küldeni a meghirdetéssel egyszerre.

7.3. Összefoglalás

Az Irinyi Kabinetben a zárthelyi dolgozatok jelenleg manuális, ímél alapú koordinációt igényelnek, amely számos hibalehetőséget rejt. Az oktatói kérésekben előfordulhatnak elírások, míg az operátori beállítások során tévedések. Egységes felület hiányában a foglaltság követése nehézkes. Ezen problémák megoldására készült a GÉZA webalkalmazás.

A dolgozat először bemutatta a vizsga mód technikai hátterét: virtualizáció, VLAN hálózatkezelés, access-control listák és ZH home könyvtárak. Ezt követően feltártuk a manuális folyamat gyengeségeit. A második fejezet a GÉZA követelményeit ismertette: LDAP bejelentkezés, BIR integráció, egyedi kérések támogatása, erőforrás-kezelés és automatikus értesítések. A harmadik fejezet a technológiai alapokat tekintette át: React, NextJS, TypeScript frontend, Better-Auth és Drizzle ORM backend, valamint LDAP, BIR és SMTP integrációk.

A negyedik fejezet az adatbázis felépítését mutatta be. Hat tábla alkotja a struktúrát: négy Better-Auth működéséhez (*user*, *session*, *account*, *verification*), kettő saját célokra (*rule*, *booking*). A BIR adatok külső nézettáblában érhetők el. Az ötödik fejezet a rendszerarchitektúrát ismertette: NextJS App Router, szerver- és kliensoldali komponensek, három szerepkör (oktató, operátor, adminisztrátor) öt erőforráshoz. A hatodik fejezet az integrációt érintette: webszerver, adatbázis és környezeti változók szükségesek. A hetedik fejezet a jövőbeli potenciált vázolta: automatizált foglalások, gépenenkénti beállítások, tanári ellenőrzés és külső rendszerekkel való integráció.

A GÉZA sikeresen valósítja meg célját: strukturált, validált környezetet teremt a vizsga mód igénylésére. A BIR integráció megakadályozza az érvénytelen foglalásokat, az LDAP kiküszöböli a külön felhasználókezelést, az e-mailek egyértelművé teszik a kommunikációt. A központi naptár átláthatóvá teszi a foglaltságot. A szerepkör-alapú jogosultságkezelés garantálja a megfelelő hozzáférést.

Az elkészült szoftver nem végleges termék, hanem alapépítmény egy hosszú távú vízió felé. A moduláris felépítés és modern technológia lehetővé teszi a bővítést: automatikus szabályalkalmazás, valós idejű követés, API-alapú kommunikáció. A jelenlegi cél a hibák minimalizálása és a továbbfejlesztés alapjainak megteremtése volt – ezt sikeresen elérte. GÉZA jelentősen egyszerűsíti a zárthelyi adminisztrációt, miközben megnyitja az utat fejlettebb megoldások felé.

Irodalomjegyzék

1 Documentation – Orcale VirtualBox https://www.virtualbox.org/wiki/Documentation [Utolsó megtekintés: 2025. december 4.]

2 Bíró https://biro3.inf.u-szeged.hu/ [Utolsó megtekintés: 2025. december 5.]

3 JetBrains: Essential tools for software developers and teams https://www.jetbrains.com/ [Utolsó megtekintés: 2025. december 5.]

4 Command Reference https://www.cisco.com/E-Learning/bulk/public/tac/cim/cib/using_cisco_ios_software/cmdrefs/access-list.htm [Utolsó megtekintés: 2025. december 4.]

5 Szegedi Tudományegyetem https://unics.szte.hu/ [Utolsó megtekintés: 2025. december 5.]

6 React Reference Overview – React https://react.dev/reference/react [Utolsó megtekintés: 2025. december 4.]

7 Next.js Docs | Next.js https://nextjs.org/docs [Utolsó megtekintés: 2025. december 4.]

8 RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol https://datatracker.ietf.org/doc/html/rfc4511 [Utolsó megtekintés: 2025. december 4.]

9 BIR | Bejelentkezés https://www.inf.u-szeged.hu/bir2/ [Utolsó megtekintés: 2025. december 5.]

10 Órarend https://www.inf.u-szeged.hu/bir2/work/index.php [Utolsó megtekintés: 2025. december 5.]

11 NIST Password Guidelines: 2025 Updates & Best Practices https://www.strongdm.com/blog/nist-password-guidelines [Utolsó megtekintés: 2025. december 5.]

12 Future-Proofing Enterprise Security: Transitioning Legacy Authentication to Modern IAM https://www.ijfmr.com/papers/2025/1/37184.pdf [Utolsó megtekintés: 2025. december 5.]

13 Getting Started: Linking and Navigating | Next.js https://nextjs.org/docs/app/getting-started/linking-and-navigating [Utolsó megtekintés: 2025. december 5.]

14 Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. https://tailwindcss.com/ [Utolsó megtekintés: 2025. december 5.]

15 Introduction - shadcn/ui https://ui.shadcn.com/docs [Utolsó megtekintés: 2025. december 5.]

16 Guides: Backend for Frontend | Next.js https://nextjs.org/docs/app/guides/backend-for-frontend#server-actions [Utolsó megtekintés: 2025. december 5.]

17 Introduction | Better Auth https://www.better-auth.com/docs/introduction [Utolsó megtekintés: 2025. december 5.]

18 Admin | Better Auth https://www.better-auth.com/docs/plugins/admin [Utolsó megtekintés: 2025. december 5.]

19 MySQL :: MySQL Documentation https://dev.mysql.com/doc/ [Utolsó megtekintés: 2025. december 5.]

20 Drizzle ORM – MySQL https://orm.drizzle.team/docs/get-started/mysql-existing [Utolsó megtekintés: 2025. december 5.]

21 erickweil/better-auth-credentials-plugin: LDAP authentication plugin for Better Auth https://github.com/erickweil/better-auth-credentials-plugin [Utolsó megtekintés: 2025. december 5.]

22 LDAP Authentication: What It Is and How It Works – JumpCloud https://jumpcloud.com/blog/what-is-ldap-authentication [Utolsó megtekintés: 2025. december 5.]

23 Neptun - SZTE / USZ https://neptun.szte.hu/ [Utolsó megtekintés: 2025. december 5.]

[24] RFC 5321 - Simple Mail Transfer Protocol https://datatracker.ietf.org/doc/html/rfc5321 [Utolsó megtekintés: 2025. december 5.]

[25] Database | Better Auth https://www.better-auth.com/docs/concepts/database#core-schema [Utolsó megtekintés: 2025. december 5.]

[26] PlantUML https://tinyurl.com/4e6bfykt [Utolsó megtekintés: 2025. december 11.] [Megjegyzés: URL lerövidítve áttekinthetőségi okokból]

[27] Normálformák | Adatbázisok gyakorlat https://www.inf.u-szeged.hu/~gnemeth/kurzusok/adatbgyak/exe/Normalizalas_web/normlformk.html [Szerző: Németh Gábor] [Utolsó megtekintés: 2025. december 5.]

[28] Next.js Docs: App Router | Next.js https://nextjs.org/docs/app [Utolsó megtekintés: 2025. december 5.]

[29] Getting Started: Project Structure | Next.js https://nextjs.org/docs/app/getting-started/project-structure [Utolsó megtekintés: 2025. december 5.]

[30] API Reference: File-system conventions | Next.js https://nextjs.org/docs/app/api-reference/file-conventions [Utolsó megtekintés: 2025. december 5.]

[31] Getting Started: Server and Client Components | Next.js https://nextjs.org/docs/app/getting-started/server-and-client-components [Utolsó megtekintés: 2025. december 5.]

[32] Next.js - shadcn/ui https://ui.shadcn.com/docs/dark-mode/next [Utolsó megtekintés: 2025. december 11.]

[33] Sonner - shadcn/ui https://ui.shadcn.com/docs/components/sonner [Utolsó megtekintés: 2025. december 11.]

[34] Installation | ilamy Calendar https://ilamy.dev/docs/getting-started/installation/ [Utolsó megtekintés: 2025. december 11.]

[35] Docker Docs https://docs.docker.com/ [Utolsó megtekintés: 2025. december 5.]

Nyilatkozat

Alulírott Vad Avar, mérnökinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Képfeldolgozás és Számítógépes Grafika Tanszék Tanszékén készítettem, mérnökinformatikus BSc diploma megszerzése érdekében.

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

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Dátum

Aláírás

Köszönetnyilvánítás

Dolgozatom nem készülhetett volna el témavezetőm, Dr. Németh Gábor nélkül, aki már az első ímélváltás óta nagyon támogatta és jó ötletnek gondolta egy ilyen rendszer kialakítását. Az ő segítsége, ötletei és szakmai meglátásai nélkül ezen rendszer talán sosem készült volna el ilyen minőségben. Türelméért és megértéséért különösen hálás vagyok neki.

Továbbá külön köszönet az Irinyi Kabinet munkatársainak, név szerint Angyal Tamás, Hemmert János, Hecskó Gabriella, Kocsorné Sebő Marianna, [HIÁNYOZNAK MÉG NEVEK?], akik többször szánták rám idejüket, hogy átbeszéljük a rendszer működését, és integrálását. A technikai hátterét a rendszernek ők biztosítják, és végsősoron ők üzemeltetik majd.

Ezen felül a rendszer előzetes átbeszélését, és az ideális vízió felállításáért köszönet illeti Goldman Júliát és Gercsó Márkot. Remélem egyszer elkészül az ideális rendszer, amit együtt kitaláltunk.