DUNAÚJVÁROSI EGYETEM

MÉRNÖKINFORMATIKUS BSC

SZAKDOLGOZAT

GYÓGYPEDAGÓGUSOK MUNKÁJÁT SEGÍTŐ WEBOLDAL FEJLESZTÉSE

Balogh Zsolt

mérnökinformatikus jelölt

A-036-INF-2023

Kivonat

A szakdolgozatom témája: Gyógypedagógusok munkáját segítő weboldal fejlesztése, amelyben egy saját webalkalmazás fejlesztéséről írok.

Szakdolgozatom elkészítése során a motivációim közé tartozott, hogy olyat alkossak, ami hasznos és segítség lehet a társadalom számára. A szakdolgozatom célja egy ingyenesen használható weboldal fejlesztése, amely segítséget nyújt a gyógypedagógusok fejlesztő munkájában. A rászorulók fejlesztéséhez a zongoratanulást használja fel, ezáltal főként enyhén értelmi sérült és tanulási zavarral rendelkezők számára készül. Az alkalmazás általános weboldal funkciók mellett a hangszertanulás egy speciális módszerét mutatja be és arra épülve egy virtuális zongorát biztosít, amely több szempontból is hasznot jelent ezen a területen.

A dolgozatomban kezdésként szót ejtek az elméleti háttérről, okokról, amiért a választott téma mellett döntöttem, illetve az alkalmazás potenciális előnyeiről és hasznosságairól. Következőkben magáról a fejlesztéséről írok, amin belül bemutatom a felhasznált technológiákat, a tervezést, a virtuális zongora és az oldal további funkcióinak implementálását. Továbbá kitérek a weboldalak ellenirányuló leggyakoribb támadásokra, azok elleni védelmekre és az általam felhasznált megoldásokra. Az elkészült funkciók és védelmi lépések után a tesztelést mutatom be, amit később a továbbfejlesztési lehetőségek bemutatása követ.

A munkám során végeredményül sikerült egy webalkalmazást készítenem, ami újnak tekinthető, hiszen kutatásaim során nem találkoztam ilyennel, valamint képes segítséget nyújtani a sajátos nevelési igényűek fejlesztésében a rendelkezésre álló virtuális zongorával.

Abstract

The topic of my thesis is: Developing a website to support the work of special needs teachers, in which I write about the development of my own web application.

My motivation for writing this thesis was to create something that would be useful and helpful to society. The aim of my thesis is to develop a free website that can be used to help in the development work of special education teachers. It uses piano learning as a tool for the development of people in need, thus it is mainly designed for people with mild intellectual disabilities and learning disabilities. In addition to general website functions, the application presents a specific method of instrumental learning and builds on it to provide a virtual piano, which is useful in this area in several ways.

In my thesis, I will start with a discussion of the theoretical background, the reasons why I chose this topic, and the potential benefits and usefulness of the application. In the following I will write about the development itself, describing the technologies used, the design, the implementation of the virtual piano and the additional features of the site. I will also cover the most common attacks that web sites are vulnerable to, the defences against them and the solutions I have used. After the completed features and protection steps, I will present the testing, followed by a discussion of further development options.

As a result of my work, I have managed to create a web application, which can be considered as a new one, as I have not encountered anything like it in my research, and it is able to help in the development of special needs children with the available virtual piano.

Tartalomjegyzék

1. Bevezetés

Napjainkban az internet fejlettsége olyan szintet ért el, hogy az emberi élet jelentős területén szinte elengedhetetlenné vált. Kezdetben a weben csak statikus oldalak voltak elérhetőek, azonban az idő haladásával ezeket leváltották a dinamikus oldalak, amelyek a felhasználókat a web aktív résztvevőjévé tették [1]. A weboldalak fejlődése szerepet játszott abban, hogy képesek legyünk az internetet több területen is felhasználni, így felgyorsítani, egyszerűsíteni és hatékonyabbá tenni az adott műveleteket. Ezen területek közé sorolhatók: az információközlés, ügyintézések, illetve az oktatás is. Edukáció esetében az internet fejlődése többek között lehetővé tette az otthoni, online tanulást és a hozzájáruló hatékonyabb, interaktív tanító weboldalak létrehozását.

Az innovatív tanulást segítő webes portálok lehetőséget adnak a gyengébb tanulási képességű személyek segítésére is. Őket a sajátos nevelési igényűek közé sorolják és képességeik fejlesztésével az oktatás egy speciális területe foglalkozik, a gyógypedagógia. A gyógypedagógus a sajátos nevelési igényű tanuló speciális igényeinek megfelelő módszereket használva fejleszti a gyerekek kognitív képességeit [2]. A sajátos nevelésűek különböző kategóriákba sorolhatók: látássérült gyermek, hallássérült gyermek, beszédfogyatékos, enyhén értelmi fogyatékos gyermek stb. [3]. Fejlesztésükhöz használt módszerek közé tartozik a zenei fejlesztés is, amely az értelmi fogyatékos, azon belül enyhén értelmi fogyatékos vagy tanulási zavarral küzdő személyek esetében használható. A fejlesztésnek effajta megközelítése hasznos, mivel a képzelet, kreativitás, figyelemkoncentráció, tartós figyelem, emlékezet és térlátás mellett még sok mást is képes fejleszteni [3].

1.1. A dolgozat célkitűzései

A 21. században elterjedtek az olyan webes tanulást segítő portálok, amelyek szöveges tartalom mellett, képekkel, videó anyagokkal, esetleg játékos feladatokkal, illetve a mai virtuális valóságot felhasználva segítik az egyén tanulását, fejlődését.

Mérnökinformatikus tanulmányomhoz kapcsolódva a szakdolgozatom témája egy ingyenes weboldal fejlesztése, amely a gyógypedagógusok munkáját segíti. Az oldal korábbi zenei tanulmányaim és még jelenleg is fennálló érdeklődésem okán a hangszeres zenetanulásra, pontosabban a zongoratanulásra épül. Az általam készített weboldallal elsősorban enyhén értelmi fogyatékos vagy tanulási zavarral küzdő személyek fejlesztéséhez tervezek hozzájárulni.

A hozzájárulás érdekében a zenetanítás kevésbé ismert színeskotta-módszerét alkalmazom, amelyet elsősorban a weboldalam célközönségének megfelelő személyek fejlesztésére használnak. A módszer nem igényel komplex gondolkodásmódot, mivel általános kotta rendszer helyett színes formákat használ, a hangok magasságát különböző színekkel, a ritmusokat pedig különböző formákkal jelöli [4]. Egyszerűségéből adódóan bármilyen képességű egyén számára segítséget tud nyújtani a hangszertanulásban, függetlenül attól, hogy rendelkezik vagy nem rendelkezik speciális szükségletekkel.

Tehát a magam által fejlesztett weboldal ezen a színeskotta-módszeren alapul. Középpontjában a színes formákra épülő virtuális zongora és kották állnak, továbbá magának a módszernek a bemutatása.

Motivációm alapjául elsősorban az szolgált, hogy olyasmit hozzak létre, ami hasznos és segítséget tud nyújtani a társadalomnak. Az interneten több hagyományos zongora is elérhető, főként szórakoztató céllal. Kis számban, de fellelhető olyan oldal is, ami a színeskotta módszert mutatja be, azonban a témában való kutatásaim során nem találtam, olyan kivitelezést, amely a kettőt ötvözné. Megfelelő és újszerű ötletként fogalmazódott meg bennem egy weboldal készítése, ami a színeskotta-módszeren alapuló virtuális zongorával, webes formában támogatja a zongoratanítást. Ezzel együtt a gyógypedagógiai fejlesztést az általa nyújtott lehetőségekkel, előnyökkel.

Jelenleg a hangszeres fejlesztéshez nagyrészt ütős hangszereket használnak, mivel egyszerűbbek, helytakarékosabbak és olcsóbbak, mint egy zongora. Utóbbiakat tekintve oldalam lehetőséget nyújt a használható eszközök kibővítésére, mivel egy valódi zongorával ellentétben a használatbavétele ingyenes. Használatához egy számítógépre vagy laptopra, illetve internetre van szükség, ami még helytakarékosságot jelent egy valódi zongorához képest. Továbbá fontos megemlíteni, hogy manapság minden iskola és háztartás rendelkezik internettel és számítógéppel. Ebből adódóan a gyógypedagógusok munkáját is megkönnyítheti olyan módon, hogy munkájuk nem feltétlen lenne helyhez kötött, esetleg esélyt adhat arra, hogy interneten keresztül is dolgozzanak, illetve így olyan iskolákban vagy háztartásban is használható lenne a módszer, ahol nem rendelkeznek zongorával.

1.2. A dolgozat felépítése

Szakdolgozatom több fejezetre oszlik, amelyek többségben a webalkalmazásom fejlesztéséhez kapcsolódnak. A dolgozatom felépítése és az egyes fejezetek tartalma az alábbiakban tekinthető meg:

Bevezetés: A bevezetés fejezetben fejtem ki a téma elméleti hátterét, dolgozatom alapját, célját, a motivációt, ami ehhez az ötlethez vezetett és a körülményeket, amik miatt az elkészült alkalmazás hasznos lehet.

Felhasznált technológiák: A szakdolgozat második fejezetén belül írok a fejlesztésnél felhasznált technológiákról és eszközökről. Az említetteket kliens- és szerveroldal szerint csoportosítottam, illetve külön kitérek a weboldalon felhasznált képek és hangfájlok elkészítéséhez használtakra is.

Tervezés: A tervezés fejezet tartalmazza az összes olyan tervezéssel kapcsolatos lépést, amelyeket implementálás előtt kell elvégezni. A fejezetben írok az oldalakhoz felhasznált erőforrások elkészítéséről, a grafikai felület, adatbázis és program tervezési lépésekről.

Elkészült grafikai felület: A negyedik fejezet tartalmazza a tervezési részben említett meghatározásoknak megfelelően elkészült grafikai felület bemutatását. A fejezeten belül szót ejtek az oldalon látható színek választásának, illetve a további megjelenéssel kapcsolatos döntéseim okairól.

Implementáció: Az implementáció fejezet tartalmazza a funkciók megvalósításának bemutatását. A fejezetben bővebben kifejtem az egyes funkciók működését a forráskód alapján. Emellett kitérek magának a programnak a felépítésére és az oldalak közötti útvonal választására készített útválasztóra.

Weboldalak védelme: A hatodik fejezet a weboldalak védelméről szól, amiben az OWASP alapítványról, valamint általuk nyújtott Top 10 listában szereplő biztonsági kockázatokról és azok elleni védelemről írok, amelyek egy kiinduló pontként szolgálnak az applikációk védelmében. Alkalmazásom biztonságának érdekében néhány, nem feltétlen a listában szereplő megoldásokat implementálok az alkalmazásomba és azoknak a kivitelezését is bemutatom.

Tesztelés és tovább fejlesztési lehetőségek: A tesztelés a fejlesztés egyik legfontosabb része, hiszen ekkor derül ki, hogy az elkészített alkalmazás megfelelően működik-e és publikálásra is alkalmas-e. A fejezetben említem meg a tesztelés menetét, eredményét és

az általam használt tesztelési technikákat. A tesztelés mellett még szót ejtek az oldal lehetséges továbbifejlesztéseiről is, amiket a jövőben megvalósítanék.

Összefoglalás: A dolgozatom legvégén, az összefoglalásban összegzem dolgozatom témáját, célját és a motivációimat. Röviden írok arról, hogy mikre tértem ki a szakdolgozatom során. Továbbá megemlítem a weboldalammal kapcsolatos elvárásokat és azok alapján végeredményül létrejött alkalmazásom jellemzőit.

2. Felhasznált technológiák

Az alkalmazás elkészítésénél arra törekedtem, hogy olyan technológiákat és eszközöket használjak, amiket a mai világban is alkalmaznak egy webalkalmazás készítése során. Választásom kliensoldalt tekintve a *HTML*, *CSS*, *Bootstrap* és JavaScript technológiákra, míg a szerveroldal esetében a *PHP*, *MySQL* és a *Gmail* által biztosított megoldásra esett. A fejlesztés folyamán még *Adobe Illustratort*, *Photoshopot*, *FL Studiot* és *XAMPP-et* vettem igénybe, mint további eszközök.

2.1. Felhasznált kliensoldali technológiák

A következőkben az alkalmazásom elkészítéséhez használt kliensoldali technológiákat mutatom be.

HTML, oldal struktúrájának meghatározója:

A *HTML* az angol HyperText Markup Language rövidítése, egy hiperszöveges leíró- vagy jelölőnyelv, amely mára a webfejlesztés egyik alapjának tekinthető. A leírónyelv az *SGML-ből* származik és a weboldalak strukturális meghatározásáért felelős [5]. A hipertext lényege, hogy különböző *HTML* fájlok vagy hivatkozások összekapcsolhatóak lesznek. Ezzel elérve, hogy az oldalon való haladás ne csak lineáris legyen, hanem egyik szövegrészről a másikra tudjunk ugrani vagy egy oldalról átlépjünk egy másikra [6].

A *HTML* a webfejlesztésnek egyik alapköve, segítségével képesek vagyunk a weboldalak felépítését alakítani. A *HTML* még lehetőséget ad az úgynevezett szemantikus web készítésére, aminek a fő előnye, hogy a böngésző képes az információk jelentésének és közöttük lévő kapcsolatok feldolgozására. Ehhez a weboldalakon megadott metaadatokra van szükség, amelyek célja, hogy segítsenek az adatok jelentésének pontos megértésében. A metaadatoknak több típusai vannak, közéjük tartozik a szemantikus címkézés is. Néhány használatos címkék, amelyeket én is használtam a weboldalam készítése során: [7]

<header>,<nav>,<main>,<article>,<footer>

További előnyei, hogy a *HTML* bővíthetőségi mechanizmusokkal rendelkezik, ilyenek például: a class attribútum, meta címkék, amelyekkel az egész oldalra kiterjedő metaadatokat vehetünk fel és „rel” mechanizmus, amit arra használhatunk, hogy az oldalon található linkeket sajátos jelentéssel lássuk el. [8]

CSS, ami az oldal stílusáért felelős:

A *HTML* későbbi fejlesztései lehetővé tették a szövegek színét és kinézetét változtató tulajdonságokat. Azonban ez azt okozta, hogy az új megjelenítést változtató elemek keveredjenek a már eddig meglévőkkel, így a nyelv elvesztette az egyszerűségét. Másik hátrányát jelentette, hogy minden oldalnak külön-külön meg kellett adni ezeket a stílusokat. [9]

A *CSS*-t az említett problémáknak a megoldására készítették. *CSS* a Cascading Style Sheets rövidítése, ami magyarul lépcsőzetes stíluslapokat jelent. Használatával megadhatjuk az oldalunk vagy abban található elemek stílusát, például: betűtípus, szöveg színe, oldal hátterének a színe. A *CSS* legnagyobb előnye, hogy külön .css kiterjesztésű fájlt hozhatunk létre, amelyet hozzá csatolhatunk a különböző oldalakhoz, valamint időt spórolhatunk, hiszen ekkor csak egy fájlt kell változtatnunk a több *HTML* fájl helyett. [9]

Stílusok megadásával kapcsolatban fontos még kiemelni, hogy egy elemre egyszerre több stílus is vonatkozhat, emiatt előfordulhat, hogy adott stílusok ütköznek egymással és megpróbálják felülírni a másikat. Ebben az esetben érvényesül a nevében rejlő Cascading kifejezés, ami arra utal, ahogyan a *CSS* értelmező priorizálja az egy elemre eső stílusokat. A szabály, hogy mindig a szűkebb hatókörrel rendelkező stílus fog érvénybe lépni [9].

Az alkalmazás fejlesztéséhez a vanilla *CSS* mellett, egy frontend keretrendszert használtam, a *Bootstrapet*. A *Bootstrap* lényege, hogy egyszerűsítse és gyorsítsa a weboldalak kinézetének elkészítését. Maga a keretrendszer előnyei nagy szerepet játszottak abban, hogy rá essen a választásom. Előnyei közé tartozik, hogy minden modern böngészővel kompatibilis, jó dokumentációval és széles közösséggel rendelkezik, bővíthető és személyre szabható. A *Bootstrap* emellett előre elkészített osztályokat, színeket, elrendezést és elemeket biztosít, amikkel egyszerűbbé és gyorsabbá teszi a fejlesztést. Az előre biztosított osztályokat szükség esetén felülírhatjuk, valamint új saját osztályokat is létrehozhatunk vanilla *CSS-el*. A keretrendszer a mobil nézet készítést is egyszerűbbé teszi az úgynevezett „mobile first” reszponzív felépítése által. [10]

JavaScript, a kliens oldali funkciók működéséhez:

A *JavaScript (röviden JS)* egy kliensoldali programozási nyelv, amelyet a böngészőkbe lévő *JavaScript-motor* értelmez és futtat le. Mára a böngészők és okostelefonok szinte mindegyike támogatja. A *JavaScriptet* két féle módon adhatjuk hozzá a weblapunkhoz, beágyazottként vagy külön fájlként csatolva. A programozási nyelvnek a jellemzői közé tartozik, hogy „case sensitive" vagyis különbséget tesz a kis- és nagybetűk között. [1]

Elsősorban a felhasználók oldalon töltött élményének a növelésére használják. Ilyen megoldások lehetnek a dinamikus frissítés, 2 és 3D grafika használata, videó lejátszás stb. Ezeket a nyelv a weblapot felépítő objektumok manipulálásával, a *DOM* manipulálásával valósítja meg [1].

A *DOM (Document Object Modell)* egy objektum modell, ami a böngésző adott *HTML* dokumentumának a struktúráját egy faszerkezetbe szervezi és ilyen formában elérhetővé teszi a *JavaScript* számára. A fastruktúrában minden pont egy adott elemet jelent. [9]

A *JS* használatának másik előnye, a benne lévő *AJAX (Asynchronous JavaScript and XML)* technika. A technika magába foglal több technológiát is, például: *HTML*, *CSS*, *JavaScript*, *DOM* és az *XMLHttpRequest* objektumot, ami lehetővé teszi a kliens és a webszerver közötti kommunikációt. Az *AJAX* képes gyorsabbá tenni egy webalkalmazást, mivel használata lehetővé teszi a megjelenített adatok frissítését és a szerverrel való kommunikációt az oldal teljes újratöltésé nélkül. [1]

2.2. Felhasznált szerveroldali technológiák

A kliensoldali technológiák után következő részben alkalmazásom szerveroldali technológiáiról ejtek szót.

PHP, használt szerveroldali programozási nyelv:

A *PHP* elsősorban webalkalmazásokhoz fejlesztett szerveroldali programozási nyelv, használatával dinamikus weboldalakat készíthetünk. Eleinte a neve *Personal Home Page Tools* volt, ilyenkor még csak egy makrógyűjteményt jelentett. A későbbi változtatások után lett *Hypertext Preprocessor* a neve és ettől kezdve vált önálló programozási nyelvé. [1]

Szintaktikája hasonló a *JavaScripthez*, használatakor a *PHP* kódot a *HTML* kódba ágyazhatjuk bele, ebből kiindulva egy *PHP* állomány a saját nyelvű kódján kívül még *HTML* kódot is tartalmaz. Kivitelezéséhez a *PHP* értelmező feldolgozza a fájlt és a *HTML* szöveg karaktereit szimpla szövegként, a *PHP* kódot pedig programkódként kezeli, majd értelmezi és végrehajtja. Az állomány lefutása után visszaküldi a lefuttatott kódok által létrejött kimenetet. A nyelv *HTML-be* ágyazhatóság mellett, adatbázis szerverrel való kommunikációra is lehetőséget ad. [1]

A *PHP* a forráskódot szerveren futtatja le és az adatokat a szerveren tárolja, ezeknek jelentős haszna van erőforrás gazdálkodás és biztonság szempontjából. A biztonsághoz azzal a tulajdonságával járul hozzá, hogy a forráskód a kliens oldali számítógépen nem jelenik meg, csak a már előbb említett kimenet. [11]

A választásom a *PHP-re* esett hiszen, az adatbázisok széles körét támogatja és egyszerűvé teszi az adatbázis műveleteket az adatbázis specifikus kiterjesztéssel vagy absztrakciós réteggel, mint a *PDO*. Továbbá hozzájárult még, hogy a *PHP* beágyazott *HTML* kódot tartalmaz, kimeneti parancsok sokasága helyett. Az okokhoz társult még, hogy ingyenes, platform független és választási lehetőséget ad a procedurális vagy objektumorientált vagy a kettő keverékének a használatára. [12], [13]

MySql, használt adatbázis-kezelő rendszer:

A webalkalmazás fejlesztéséhez szükségem volt még egy adatbázis-kezelő rendszerre is, hogy a felhasználókkal és más egyéb dolgokkal kapcsolatos adatokat tárolni, illetve műveleteket tudjak végezni velük. A választásom a *MySql-re* esett.

A *MySql* egy relációs adatbázis-kezelő rendszer, ami lehetővé teszi számomra, hogy adatokat tároljak, manipuláljak, lekérdezzek és még más egyéb adatkezeléssel kapcsolatos műveletet végezzek el. *MySql* ahogyan a neve is sugallja, *SQL-t* használ, akárcsak a többi nagyobb adatbázis kezelők. Az *SQL-ben* használt szavak értelmes, adatkezelő műveleteket jelentő angol szavakból állnak. [14]

A többi nagyobb adatbázis-kezelővel szembeni előnyei közé tartozik, hogy ingyenesen használható, nyílt forráskódú, azonban teljesítményében és tudásában képes felvenni a versenyt a kereskedelmi szoftverekkel is [14]. Emellett *PHP-ban* is van lehetőség a használatára, ezért webes alkalmazások készítésénél az egyik legtöbbet használt adatbázis-kezelőnek számít [14]. Adatbázis-kezelésnél lehetőség van grafikus

eszközök használatára, ilyen lenne a *PHPMyAdmin*, amelyet *PHP* nyelven írtak. A saját munkám során ezt használtam.

Gmail, a levelezést használó funkciók megvalósítója:

Az alkalmazás egyes funkcióihoz szükséges egy levelező szerver. A lokális fejlesztés során a *Gmail* által nyújtott lehetőséget használtam. A működéséhez készítettem egy külön *Gmail* fiókot, amin keresztül az oldal adott funkciói képesek elküldeni a megfelelő leveleket.

2.3. További felhasznált eszközök

A szakdolgozatom elkészítéséhez szükségem volt még képekre és hangfájlokra, hogy növeljék a felhasználói élményt, illetve mivel az alkalmazás a zenére épül a hangfájlok elengedhetetlenek voltak.

Adobe Illustrator és Photoshop, a használt képszerkesztők:

A weboldal reprezentációjával kapcsolatban szükségesnek tartom a képek használatát, a felhasználói élmény növelése és a magyarázatok jobb megérthetősége érdekében. A képeknél törekedtem a saját magam által készítettek használatára, emiatt képszerkesztőkre volt szükségem. A választásom az Adobe csomagba lévő *Illustratorra* és *Photoshopra* esett.

FL Studio, a hangfájlok készítésére használt program:

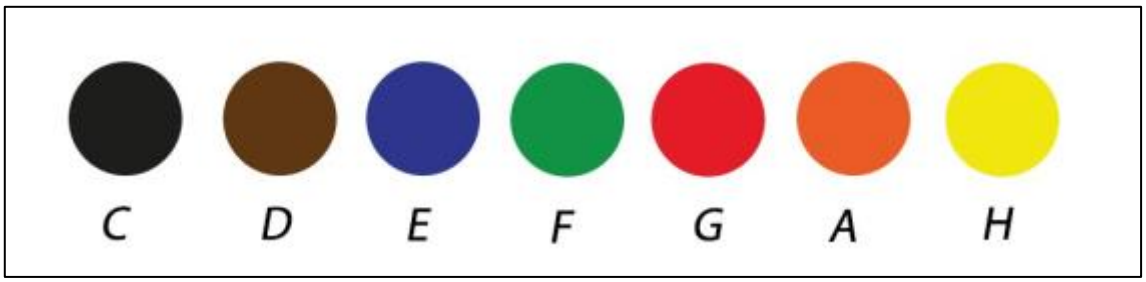
A hangfájlok *FL Studioban* készültek. A választás elsősorban azért esett erre, mivel sajnos interneten nem találtam megfelelő, ingyenesen felhasználható hangfájlokat. Ebből adódóan kénytelen voltam saját hangfájlok elkészítésére és felhasználására.

XAMPP, a lokális fejlesztéshez használt technológia:

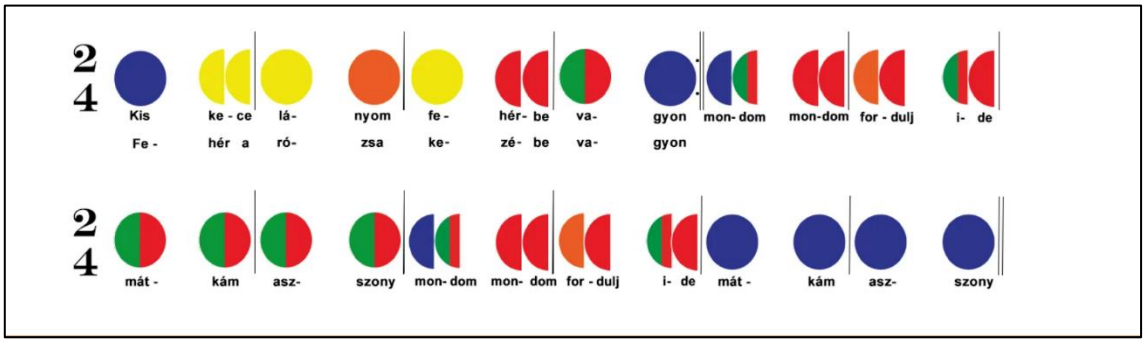
Az alkalmazás fejlesztése localhoston történt. A *PHP* mellett szükségem volt még webszerverre és adatbázis-kezelőre. Ennek érdekében *XAMPP* csomagot használtam, ami ingyenesen telepíthető és tartalmazza az előbb felsorolt számomra szükséges dolgokat, így nem kellett őket külön egyesével telepítenem és konfigurálnom [15]. A csomag még biztosítja a *PhpMyAdmin-t*, ami megkönnyíti az adatbázisok és táblák kezelését.

Fontos kiemelni, hogy az *XAMPP* használata csak webalkalmazások fejlesztésére és tesztelésére lett kitalálva. Ennek az oka, hogy néhány biztonsági beállítás alapértelmezetten ki van kapcsolva, valamint egyéb biztonsági szempontból fontos

dolgok hiányoznak, így a csomagot kifejezetten csak a fejlesztési szakaszra használtam. A weboldal publikálásához a későbbiekben pedig egy webtárhely szolgáltatást vettem igénybe. [16]



1. ábra: A színeskotta alap hangjainak jelölése ([4] alapján saját szerkesztésű ábra)



2. ábra: Kis kece lányom népdal kottája (saját szerkesztésű ábra)

3. Tervezés

A webalkalmazás elkészítéséhez nem csak fontos, de ellengethetetlen lépés az alkalmazásnál használt elemek elkészítése, valamint magának a weboldal felépítésének és működésének a megtervezése. A tervezési fázishoz tartozik például: a grafikus felület és az adatbázis megtervezése, illetve a szoftver specifikációnak a meghatározása.

3.1. Használt elemek elkészítése

A mai világban kevés olyan weboldal található, ami nem rendelkezne képekkel, hangfájlokkal vagy egyéb olyan multimédia elemekkel, amelyek a felhasználói élményt növelnék. Szakdolgozatom esetében a látogatói élmény növelése érdekében képeket és hangfájlokat használtam.

Képek felhasználásával a fő célom, hogy érthetőbbé tegyem a felhasználók számára a zongoratanításhoz használt módszer magyarázatát. Emellett a virtuális zongora billentyűin látható karikákhoz és a kottákhoz is képeket használtam. A képállományok elkészítése már említett *Adobe Illutratorban* és *Photoshopban* történt. A színeskotta-módszer színes karikái és formái az *Illustrator* segítségével készültek el. A kották pedig a már elkészült karikáknak *Photoshopban* való összerakásával jöttek létre. Az ábrák eleinte *PNG* formátummal rendelkeztek, amelyet később *WebP* formátumba konvertáltam át, az oldal gyorsaságának fokozása érdekében. Az oldalon megtalálható képekre példaként fentebb az 1. és 2. ábra szolgál.

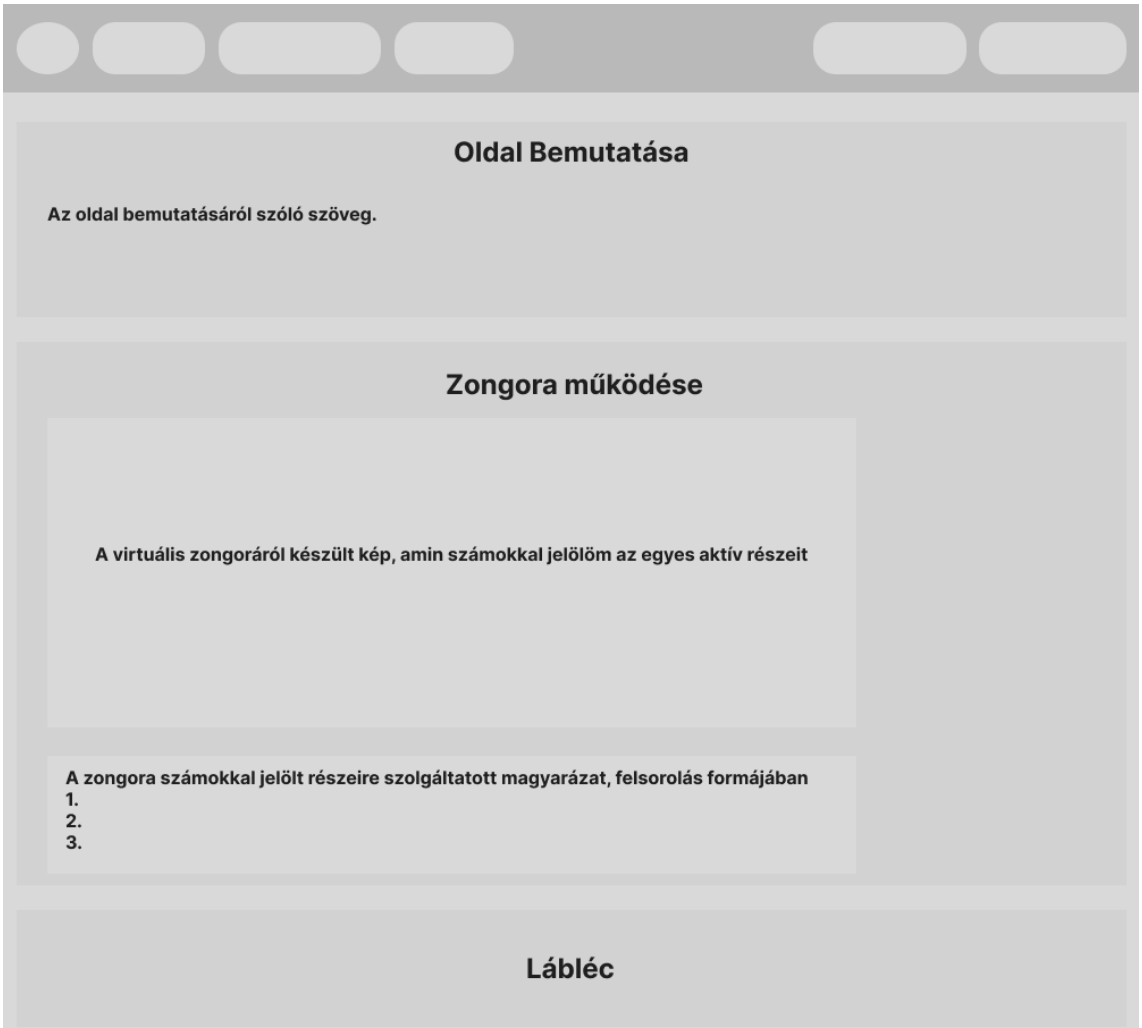
Webalkalmazásom zenére épülése okából, elengedhetetlen a hangfájlok használata. Elkészítésükhöz *FL Studiot* használtam, segítségével külön fájlokba mentve készültek el a zongora billentyűinek különböző hangjai és az oktatásra felhasznált népdalok.

A weboldalon található módszert magyarázó szövegekhez és képekhez a Nemzeti Köznevelési Portál, speciális színeskotta fejezetét vettem segítségül [4]. A hangfájlok és képek készítéséhez még megemlíteném, hogy két ismerősömtől segítséget kaptam, mivel ők hobbiként használták már ezeket a programokat sokkal jártasabbak voltak nálam bennük, így a képeket és hangfájlokat közös erővel készítettük el.

3.2. Grafikus felület tervezése

A grafikus felület megtervezésénél folytonosan szem előtt tartottam, hogy a webalkalmazás főként enyhén értelmi fogyatékos és tanulási zavarral rendelkező gyerekek fejlesztésére készül. Ennek érdekében a designt egyszerűnek és jól átláthatónak terveztem, hogy a pedagógusok számára egyszerű és egyértelmű legyen az oldal használata. A gyerekek számára pedig ne legyenek zavaró vagy figyelemelterelő elemek.

Elkészített alkalmazásom több oldalból áll, a főbb oldalak: főoldal, színeskottáról oldal, zongora oldal, elfelejtett jelszó visszaállítás oldal, valamint a regisztrációnak és a bejelentkezésnek is külön oldalakat szántam. Az említett oldalaknak a vázlatait Figmában készítettem. A főoldal vázlata alább található (3.ábra), a további oldalak vázlatait szemléltető ábrák pedig az első számú mellékletben szerepelnek (21., 22., 23., 24. ábrák).



3. ábra: Főoldalnak a vázlata (saját szerkesztésű ábra)

Oldal felépítések szempontjából arra törekedtem, hogy adott oldalnak a tartalma egyértelmű, jól átlátható, értelmezhető és lényegre törő legyen. Ahogyan az egyes vázlatokon is látható a tartalom részekre, fejezetekre lett bontva és az oldalaknak nagy területét foglalja el. Az oldalakon látható fejléccel szemben elvárás, hogy a felhasználó bejelentkezett státuszától függően változnia kell. Amikor be van jelentkezve, a fejléc jobb oldalán a regisztráció és a bejelentkezés gomb helyére a felhasználónév és egy ikon kerül, amivel ki tud jelentkezni a felhasználó (4.ábra).

4. ábra: A fejléc vázlata, ha a felhasználó be van jelentkezve (saját szerkesztésű ábra)

Az egyszerűségnek a használható gombok kinézetére is vonatkoznia kell, hogy passzolhasson, egy összhangot alkosson az oldal további részeivel. Általában érdemes és a felhasználói élmény növeléséhez is hozzájárul, ha jelezzük a kliensnek, hogy a főoldalak közül éppen melyiken tartózkodik. A gombok esetében még előnyös valamilyen egyszerűbb effekttel jelezni a működőképességüket.

A weboldal felületek megfelelőségében nagy szerepet játszanak a használt színek. Én esetemben a felhasználandó színeknek olyanokat kell választanom, amelyek a szöveg láthatóságának megfelelnek, egyszerűbbek, nem túl rikítóak, mégis élénkebbé teszik az oldalak kinézetét. Ezzel is tetszetősebbé téve az oldalt a gyerekek számára.

A weboldalt a korszerű követelményeknek megfelelően reszponzívra terveztem elkészíteni. A vázlatok, azonban csak asztali nézetről készültek, mivel a telefonos nézeten nincsen annyi lehetőség elemek elrendezésére, emiatt nem tartottam fontosnak a mobil felületek külön megtervezését. A reszponzív megjelenéshez a Bootstrap által biztosított reszponzív megoldások felhasználását választottam.

3.3. Adatbázis felépítése

A webalkalmazások működéséhez manapság elengedhetetlen az adatbázisok használata. Segítségükkel nagy mennyiségű adatot tudunk tárolni biztonságosan, rendezetten, úgy, hogy többen is használhatják egy időben. Következőkben az adatbázis tervezéséről, alkalmazásom adattábláiról és a közöttük lévő kapcsolatok által létrejött sémáról írok.

3.3.1. Adatbázis tervezése

Adatbázis tervezése egy több lépésből álló folyamat, amelynek a nehézsége nőhet, az alkalmazásnak a komplexitásától. A tervezésnek a lépései:

1. **Adatbázis céljának a meghatározása:**

A tervezés első lépéseként a készülő adatbázisnak a céljait kell megfogalmazni, milyen adatokat akarunk tárolni és mi a célunk a tárolt adatokkal. Ezeket követően még meghatározásra kell, hogy kerüljön az adatok további részletei is. Például a hosszuk, típusuk és egyéb más tulajdonságaik. [17]

Esetemben az adatbázis és a tárolni kívánt adatok célja a felhasználók, a tanulható dalok és az egyes funkciók működéséhez szükséges adatok tárolása.

2. **Adatbázist felépítő táblák meghatározása:**

Miután meghatároztuk az adatbázis célját és a tárolásra kerülő adatokat, következő lépésként az adatokat csoportosítani kell témakörök szerint. A rendszerezés által létrejött csoportok fogják jelenteni a táblákat. Alkalmazásom esetében három témakörre osztottam fel az adatokat, emiatt három táblát kellett készítenem. [17]

3. **A táblákban szereplő mezők meghatározása:**

A harmadik lépésként a meghatározott táblák mezőit kell megadni, azok neveit. A neveknek érdemes beszédesnek lenniük, ezáltal egyértelműbbé válik, hogy melyik mező milyen adatot őriz. A táblák mezőire jellemző, hogy közülük egy a tábla rekordjainak egyértelmű azonosításáért felel. Ezt a mezőt nevezzük elsődleges kulcsnak, amire jellemző, hogy mindig egyedi értékkel rendelkezik, nincsen kettő megegyező értékű rekord benne és általában minden új rekord esetében az értéke egyel növekszik. [17]

Adatbázisom tábláinál törekedtem a fentebb említett beszédes mező nevekre, illetve mindegyik tábla rendelkezik egy azonosítóval, amely az elsődleges kulcs szerepét tölti be. A táblákról bővebben az adatbázis táblák alfejezetben ejtek szót.

4. A táblák közötti kapcsolatok meghatározása:

Miután meghatároztuk a táblákat és mezőkkel töltöttük fel, már csak a táblák közötti kapcsolatoknak a meghatározása maradt hátra. A kapcsolatok kötik össze a különböző táblákat, használatuk segítséget nyújthat a lekérdezésekkel való adatkinyerésben. A webalkalmazásom táblái közötti kapcsolat a séma megtervezése alfejezetben elhelyezett ábrán tekinthető meg. [17]

3.3.2. Adatbázis táblák

Elkészített adatbázisom három táblát tartalmaz, amik tárolják a felhasználókkal kapcsolatos adatokat, az alkalmazás virtuális zongoráján tanulható dalok adatait, valamint létezik egy külön tábla az elfelejtett jelszó funkcióhoz szükséges adatok tárolására. A táblák felépítését az alábbiakban mutatom be.

Users tábla:

A `users` tábla tartalmazza a regisztrációkor megadott felhasználói és egyéb velük kapcsolatos adatokat, amik adott funkciók működéséhez szükségesek. A tábla elsődleges kulcsa az `id`, amely minden új felhasználó felvételekor egyel növekszik. A `vkey` mező tartalmazza az adott kliensnek a megerősítő kódját, amivel megerősítheti a fiókját, a `verified` mező pedig azt jelzi, hogy elvégezte-e a megerősítést. A tábla tartalmazza még a fiók létrejöttének a dátumát, valamint a dátumot ameddig a fiók megerősítése elvégezhető. A megerősítés határidőn túli halasztása esetében az adott fiók úgymond törlésre kerül és a `deleted` mező 1 értéket vesz fel. A táblában található egy idegen kulcs is a `song_id`, ami a `Songs` táblára mutat és azt jelöli, hogy a felhasználó melyik dalnál tart. A tábla felépítése az 1. táblázaton látható.

1. táblázat: Users tábla

Név	Típus	Kötelező	Tartalom
id	int(11)	igen	Elsődleges kulcs (PK)
username	varchar(30)	igen	Felhasználó neve
password	text	igen	Felhasználó jelszava kódolt formában
email	varchar(100)	igen	Felhasználó email címe
vkey	text	igen	Fiók megerősítő kulcs
verified	tinyint(1)	nem	Megerősítette-e a fiókját
creation_date	date	igen	Fiók létrehozásának a dátuma
expiry_date	date	igen	Fiók megerősítés határideje
deleted	tinyint(1)	nem	Törlésre került-e a fiókja
song_id	int(11)	igen	Adott dal, ahol a felhasználó tart (FK)

Songs tábla:

A `Songs` tábla a népdalok adatait és egy azonosítót, a `song_id`-t tartalmazza. A táblában található mezők a dal címét, illetve a kottájának és hangfájljának az elérési útját tárolja. A `Songs` táblát a 2. táblázaton tekintheti meg.

2. táblázat: Songs tábla

Név	Típus	Kötelező	Tartalom
song_id	int(11)	igen	Elsődleges kulcs (PK)
song_title	varchar(50)	igen	A dal címe
song_sheet_path	text	igen	A kotta kép elérési útja
song_file_path	text	igen	A dal hangfájljának elérési útja

Resetpassword tábla:

A webalkalmazás lehetőséget ad a felhasználónak, hogy ha elfelejtette a jelszavát, akkor email címének a megadásával meg tudja változtatni. A `resetpassword` tábla eme funkciót teszi lehetővé. Azonosítója a `ResetPwdID`. A tábla ezenkívül tárolja a jelszócseréhez megadott emailt és egy lejárati dátumot, ameddig érvényes a jelszócsere kérelem. Tartalmaz még egy `ResetPwdToken` mezőt, ami a felhasználó azonosításához

kell, segítségével ellenőrzi az alkalmazás, hogy a megfelelő felhasználó végzi el a jelszócserét. A táblának a szerkezetét a 3. táblázat mutatja.

3. táblázat: Resetpassword tábla

Név	Típus	Kötelező	Tartalom
ResetPwdID	int(11)	igen	Elsődleges kulcs (PK)
ResetPwdEmail	text	igen	Jelszócserét kért email cím
ResetPwdToken	longtext	igen	Jelszócsere megerősítő kódja
ResetPwdExpDate	datetime	igen	A jelszócsere határideje

3.3.3. Séma megtervezése

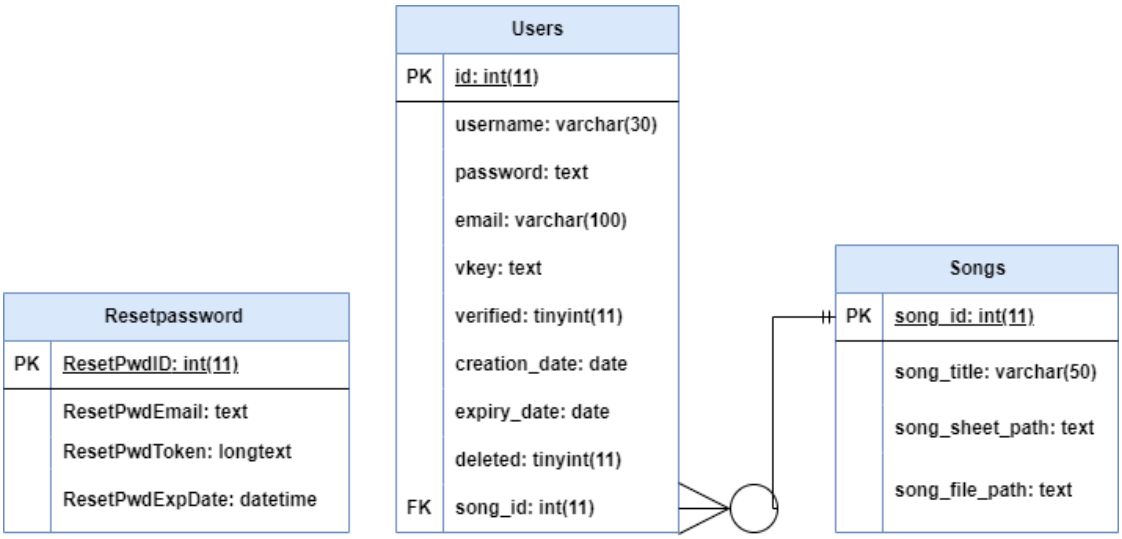
A táblák között kapcsolatok állhatnak fent, amiket a séma tervezése során szintén figyelembe kell venni. A kapcsolatok gyakorlatban egyik táblának az elsődleges kulcsának a másik táblának az idegen kulcsával való összekapcsolásával állnak elő. Ezeknek több típusa van.

Egy az egyhez kapcsolat (1:1): Erről a kapcsolatról, akkor beszélünk, ha az egyik (A) táblának egy egyede, a másik (B) táblának csak egyetlen egy egyedéhez kapcsolódik és ez fordítva is igaz a kapcsolatra. [14]

Egy a többhöz kapcsolat (1:n): Az egy a többhöz kapcsolat az egyik leggyakrabban használt típus. Ez a kapcsolat típus akkor fordul elő, ha „A" táblának bármelyik egyede több „B" egyedtípusú egyedhez is kapcsolódhat, míg a „B" egyedei csak egy „A" táblában lévő egyedhez kapcsolhatók [14].

Több a többhöz kapcsolat (n:m): A több a többhöz kapcsolat esetében mindkét táblának az egyedei a másik táblának több egyedéhez is kapcsolhatók. „A" egyedtípus egyedei több „B" egyedtípusú egyedhez kapcsolhatók és „B" egyedei is több „A" egyedtípusú egyedhez kapcsolódhatnak. [14]

Az adatbázis sémájának elkészítéséhez, szükség van az elkészült, pontos, mezőkkel feltöltött táblákra, majd pedig az előbb felsorolt, adott esetnek megfelelő kapcsolattal összeköthetjük őket. Saját alkalmazásomnál összesen három tábla áll rendelkezésre, ebből adódóan az adatbázis sémám eléggé egyszerű felépítésű. Az elkészített adatbázisomnak a sémája az 5. ábrán látható.



5. ábra: Az adatbázis sémája (saját szerkesztésű ábra)

3.4. Szoftver specifikáció

A szoftver specifikáció az a szakasz, amikor a szoftverrel szemben elvárt követelményeket és elvégzendő feladatait kell megfogalmaznunk. Alkalmazásom esetében az általános elvárások mellett a következő funkciók kellenek, hogy rendelkezésre álljanak:

- regisztráció
- bejelentkezés
- fiók megerősítés
- elfelejtett jelszó funkció
- inaktív felhasználók törlése
- virtuális zongora

Általános elvárások:

Általános elvárások közé tartozik, hogy rendelkeznie kell valamilyen szintű biztonsággal és megfeleljen a felhasználói igényeknek. A biztonság érdekében megkötéseket lehet alkalmazni. Közéjük tartozhat a beviteli mezők megkötései, amik a legtöbb esetben a felhasználó által megadott adatokra vonatkoznak. Ilyenek például: jelszó esetében tartalmaznia kell legalább egy nagy, egy speciális és egy kis karaktert, valamint minimum nyolc karakterből kell állnia.

Regisztráció és bejelentkezés:

A mai weboldalaknak alapvető funkciói közé tartozik a regisztráció és bejelentkezés. Az alkalmazásnak biztonságosan kell tárolni a felhasználók adatait regisztrációkor, emiatt a jelszavakat kódolva kell tárolni az adatbázisban. A bemeneti adatoknál elvártak közé tartozik, hogy minden felhasználónévnek különbözőnek kell lennie. A regisztráció email cím bemeneténél megfelelő email formátumot kell megadni, a jelszó megadására kettő bemenetet kell biztosítani és a megadott értékeiknek meg kell egyezniük. A megadott jelszó esetében elvárás, hogy megfelelő erősségű legyen, legalább nyolc karakter hosszú és a kis karaktereken kívül tartalmazzon legalább egy nagy, egy speciális és egy numerikus karaktert is.

A regisztrációt követően a felhasználónak be kell tudnia jelentkezni a regisztrációkor megadott felhasználónév és jelszó párossal. Hibába ütközés során az alkalmazásnak jeleznie kell a felhasználó számára a hibás bemenetekből fakadó hibaüzeneteket.

Fiók megerősítés:

A regisztráció után a felhasználónak meg kell erősítenie a fiókját emailben kapott linken keresztül. A fiók aktiválásának a fontossága abba nyilvánul meg, hogy ennek köszönhetően csak olyan emberek használhatják az alkalmazás által nyújtott zongorát, akik megerősítették a fiókjukat és ténylegesen használnák.

Elfelejtette a jelszavát funkció:

Az elfelejtett jelszó funkció egy másik olyan lehetőség, amely egy szinte elengedhetetlen szolgáltatásnak minősül. Lehetővé teszi a regisztrált személy számára, hogy lecserélje a jelszavát emailben kapott link segítségével, abban az esetben, ha elfelejtette és nem tud bejelentkezni.

Inaktív felhasználók törlése:

A felhasználóknak meg kell erősíteniük a fiókjukat, a zongora funkció használatának érdekében. A megerősítés határidőn túl való halasztásakor, az alkalmazás úgymond törli, inaktívvá teszi az adott fiókot. Előfordulhat, hogy az adott személy akarata ellenére nem tudta aktiválni a fiókját (áramkimaradás, internet hiánya), emiatt a tényleges fiók törlés helyett az adatbázisban egy külön mezőt kell szolgáltatni, ami az eltávolítás

megtörténését tárolja. A művelet ilyen fajta megoldása lehetőséget ad a felhasználóknak, hogy újraaktiválják a fiókjukat.

Virtuális zongora biztosítása:

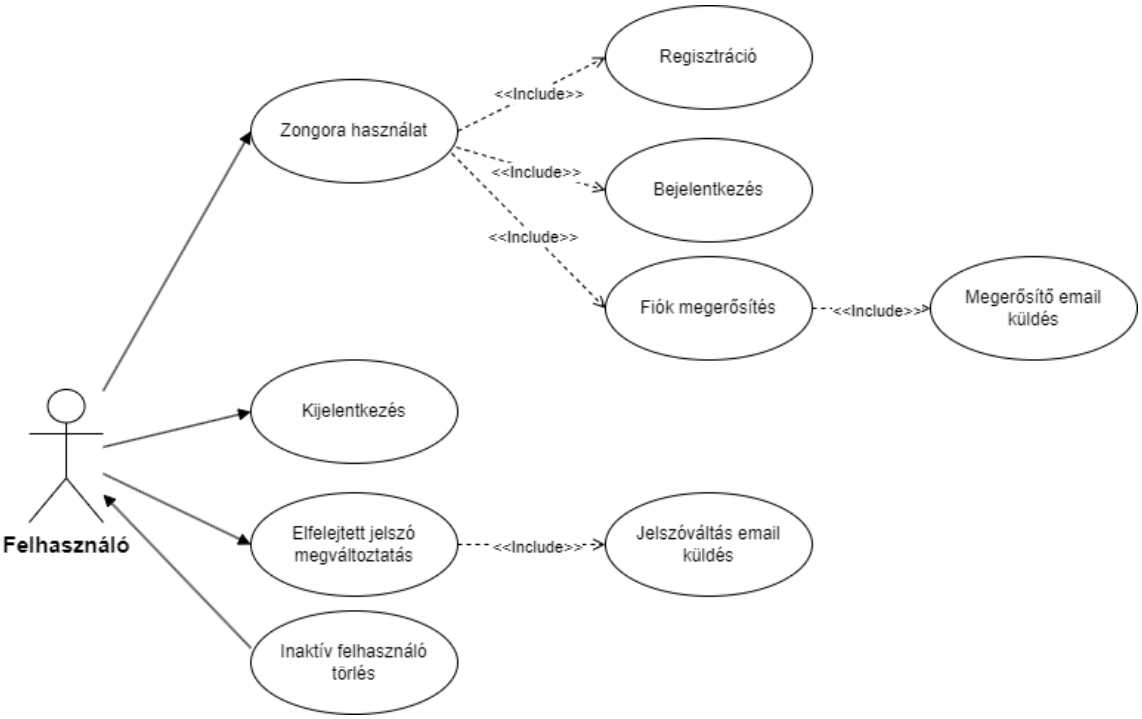
A virtuális zongora a webalkalmazásnak a legfontosabb funkciója, hiszen erre épül az egész projekt. Ebből adódóan a funkciót feltételekhez kell kötni: a zongorát használni akaró személynek kötelezően be kell regisztrálnia és jelentkeznie az oldalra, valamint még megerősített fiókkal kell rendelkeznie. A használathoz szükséges feltételeket kiegyenlítve a zongorának élethű hangot kell szolgáltatnia a felhasználónak, lehetővé kell tennie a dalok közötti váltást. A látogatók segítése érdekében ajánlott lehetőséget adni, a felhasználó számára, hogy meghallgathassa az eredeti dalt, így lehetővé téve, hogy valamihez viszonyítani tudja a saját játékát.

Kijelentkezés:

Az alkalmazás rendelkezik bejelentkezés funkcióval, ami azt vonja maga után, hogy egy kijelentkezés funkcióval is rendelkeznie kell az oldalnak, ugyanis természetes, hogy a felhasználó miután befejezte az oldal használatát, ki is jelentkezne, ezzel megakadályozva, hogy a számítógépéről mások is hozzáférhessenek a fiókjához.

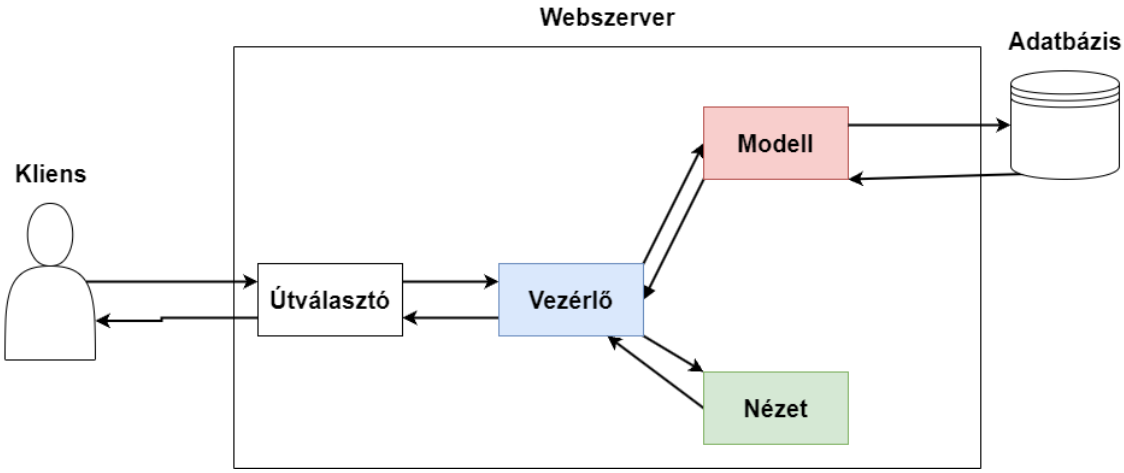
3.5. Program terv

A program tervezéséhez különféle terveket és diagrammokat készíthetünk. Alkalmazásom tervezése során egy használati eset modellt készítettem. A modell a jövőbeli felhasználó és az alkalmazás funkciói közötti interakciót ábrázolja. Az ábrán látható, hogy a specifikációk alfejezetben már felsorolt funkciók alkotják azokat a használati eseteket, amelyek előfordulhatnak a kommunikáció során (6. ábra). A nyilak pedig a kommunikáció irányát jelzik.



6. ábra: A használati eset modell (saját szerkesztésű ábra)

A programot az MVC minta szerint tervezem elkészíteni. A minta lényege, hogy a programot három részre osztjuk. A nézetre, modellre és vezérlőre. Ebből adódóan a mintának az előnyei közé tartozik, hogy a jövőbeli javításokat, fejlesztéseket könnyebben el tudjuk végezni, adott részeket külön tesztelhetjük és egy időben fejleszthetjük. [18] Az alkalmazásom említett felépítése a 7. ábrán található.



7. ábra: Az alkalmazás felépítése (saját szerkesztésű ábra)

Az ábrán a kliens jelenti az oldalt használó személyt. Első lépésben, amikor megpróbálja elérni az oldalt egy kérést küld a böngészője a webszervernek, ami a kérést

feldolgozza és egy választ küld vissza számára. A kérés feldolgozását és a megfelelő vezérlőnek irányítását az útválasztó végzi el.

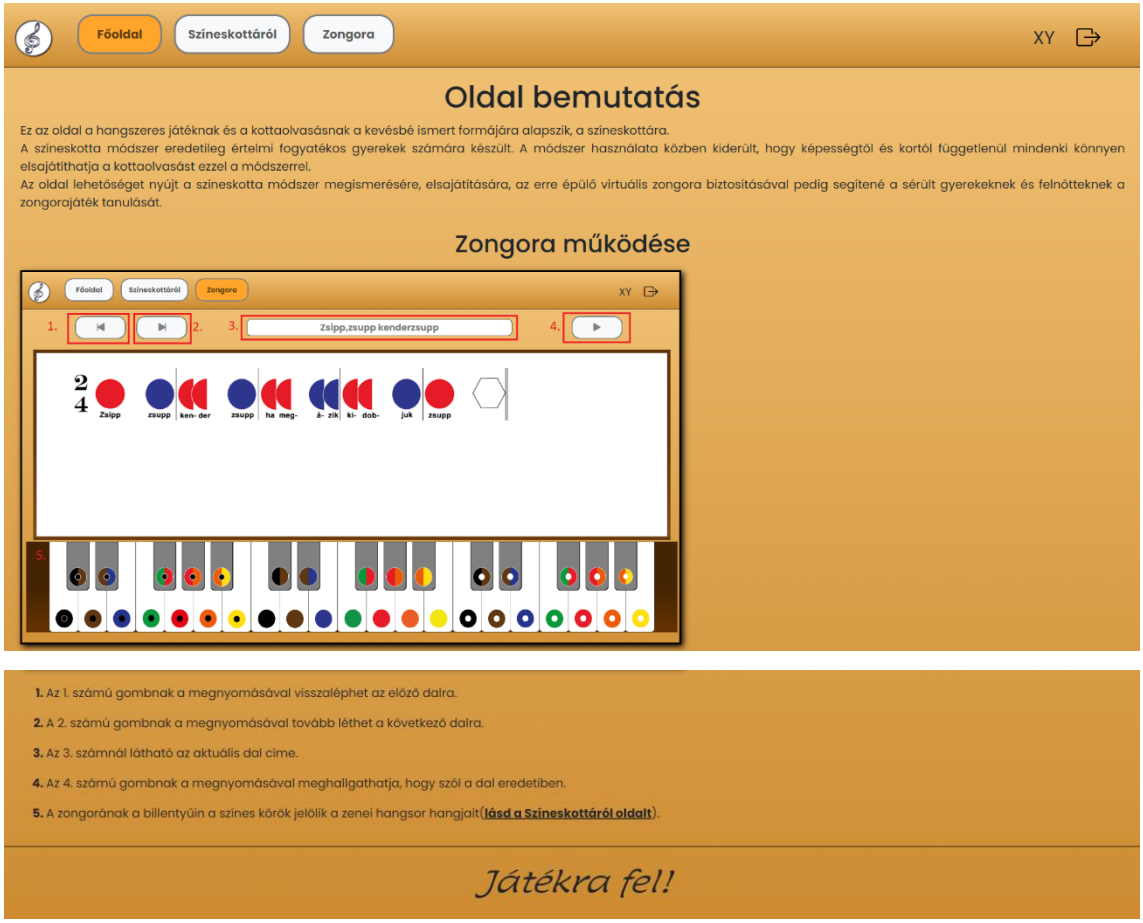
A vezérlő feladata, hogy az adott események megvalósuljanak, a megfelelő felület jelenjen meg és kapcsolatot alkosson a nézet és a modell között [18]. A működése során fogadja a kéréseket az útválasztótól majd, ha adatbázissal kapcsolatos műveletekre van szükség, akkor továbbítja a modellnek, ha nem akkor pedig a nézetnek.

Az MVC mintának egyik fontos jellemzője, hogy adatbázissal csakis a modell tud kommunikálni, ezzel egyféle egyszerűséget és biztonságot nyújt. A modell feladata az adatbázissal való kommunikáció és az összes adathoz kapcsolódó logikának a megvalósítása, amik közé tartozik az adatbázisban szereplő új adatok felvétele, a meglévők módosítása és törlése [18].

Miután a modell és adatbázis közötti kommunikáció lezárult a folyamat visszakerül a vezérlőhöz, ha lekérdezésről volt szó, akkor a megfelelő lekérdezett adatokkal. Ezután a kontroller átadja a lekérdezett adatokat a nézetnek, ami megfelelő grafikai felület előállítására és annak vezérlőhöz való visszaküldésére szolgál. Fontos szabályok, amik a nézetre mondhatóak, hogy nem tartalmazhat bonyolult logikát, minél egyszerűbbnek kell lennie és csakis a vezérlővel kommunikál, ő rajta keresztül kapja meg a modell által lekérdezett adatokat. Miután a vezérlőhöz visszakerült az előállított kimenet, azt visszaküldi az útválasztón keresztül a klienshez. [18]

4. Elkészült grafikai felület

A grafikai felület elkészítése során igyekeztem a tervezéskor megfogalmazott követelményeknek és elvárásoknak megfelelően megvalósítani az oldalak kinézeteit. Az elkészült oldalakat prezentáló ábrák alább (8. és 9. ábrák), valamint a második számú mellékletben tekinthetők meg (25.,26.,27. ábrák).



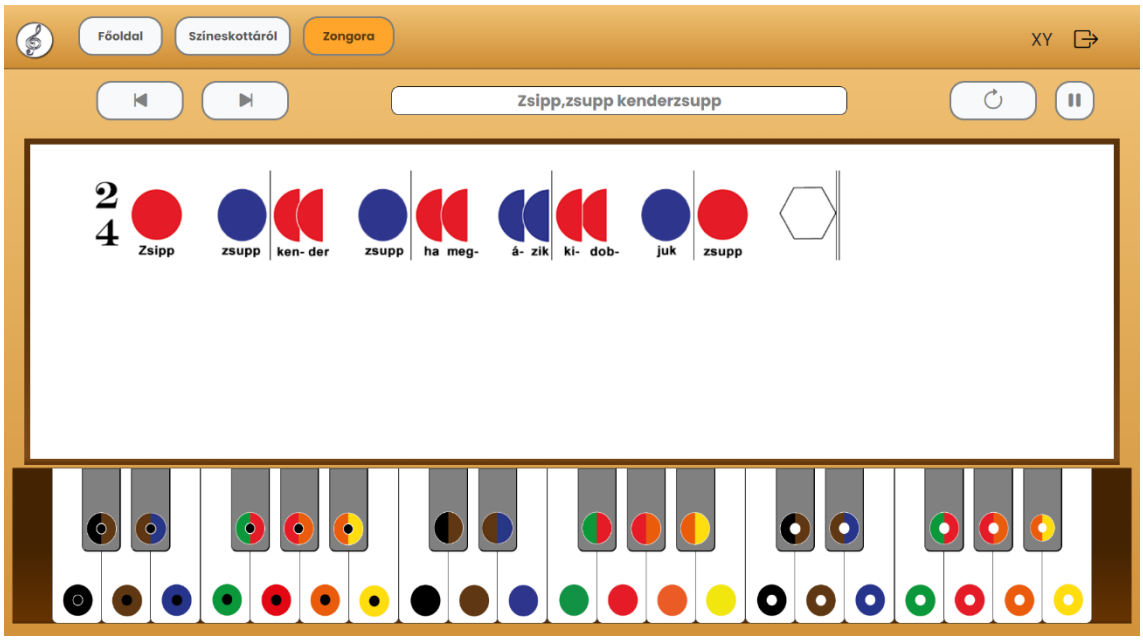
8. ábra és 9. ábra: Az elkészült főoldal kinézete (saját szerkesztésű ábra)

Az oldalak felépítésénél adott weblapnak a tartalma középpontban van, részekre bontva a jobb átláthatóság érdekében. A színeskottáról oldal esetében fontosnak tartom újra megemlíteni, hogy a kutatásaim során talált, Nemzeti Köznevelési Portál anyagait vettem alapul a módszer leírásához [4]. Ezt az oldalon levő leírás végén is kiemeltem. A fejlécnek elkészült mindkettő típusú megjelenése. A kijelentkezés ikonhoz a *Bootstrap* hivatalos oldalán nyújtott ikonok közül választottam egyet, mellette pedig látható a bejelentkezett felhasználó neve.

A gombok elkészítéséhez a *Bootstrap* egyes osztályait használtam alapul és kiegészítettem a saját magam által megírt osztállyal. Eredményként egy lekerekített, egyszerű kinézetű stílust készítettem a gomboknak. A tervezésnél meg lett határozva, hogy valamilyen módon jelezni kell a felhasználónak, hogy a főoldalak közül éppen melyiken van. Megoldása érdekében annak a fejlécben található gombnak a színe változik, amelyiknek éppen az oldalán van a felhasználó. Emellett a tervezésnél megfogalmazott egyszerűbb kurzor rávitel effektet is elkészítettem, ami a gomb háttérszínének változásában és fehér árnyékának megjelenésében valósul meg.

A választott színek esetében a meghatározás úgy szólt, hogy egyszerű, ne rikító színek legyenek, mégis élénkebbé tudják tenni az oldalak kinézetét, illetve a szövegek is jól láthatóak legyenek. Az oldal fő színének egy világos barnából kissé sötétebb barnába való átmenetet választottam. A választásom okául szolgált, hogy jól illik a zongoratanításhoz használt színeskotta-módszer színeihez, nem egy zavaróan rikító szín, emellett akár a gyerekek tetszését is képes elnyerni. A szövegek színénél maradtam az alap feketénél, mivel illik a háttérhez és olvashatóság szempontjából is megfelelő. A gombok aktív és kurzort rávitt állapotának egy narancssárga színt választottam. Elsősorban azért, mert úgy gondoltam a narancssárga szín jobban ki tudná emelni a gombokat a jelenlegi a háttérből.

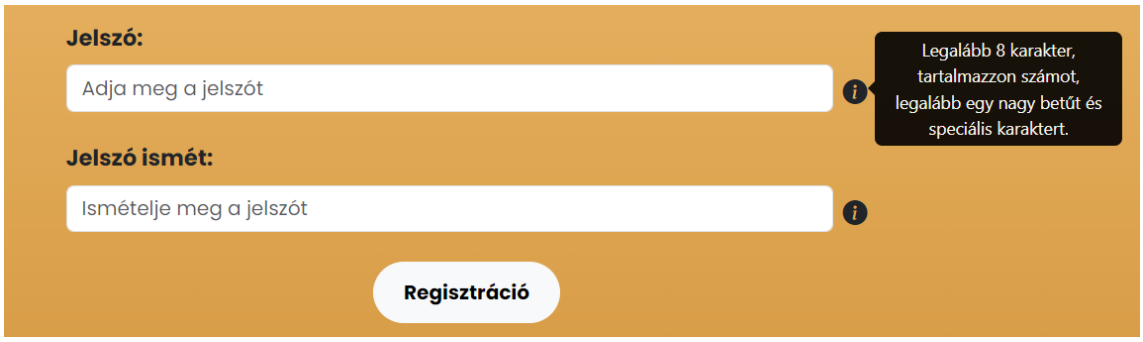
A zongorának a méretét nagyra készítettem el, valamint a színeskotta módszer karikáit is elhelyeztem a billentyűkön, hogy a sérült gyermek könnyebben tudjon tájékozódni, könnyebben tudja lenyomni a megfelelő billentyűt és a koncentrációban is segítsen neki. A zongorát telefonos méretre nem készítettem el, mivel kisebb képernyőn a billentyűk és a kották is kisebbek lesznek, ez pedig csak nehezítené a zongorának a használatát a diák számára. A zongora minimum 1024x768-as felbontásban és afelett érhető el, ami azt jelenti, hogy a legkisebb képernyő méret, amin használható az egy tablet elfordított képernyője. Az elkészült virtuális zongora oldal felülete a 10. ábrán látható.



10. ábra: A zongora kinézete (saját szerkesztésű ábra)

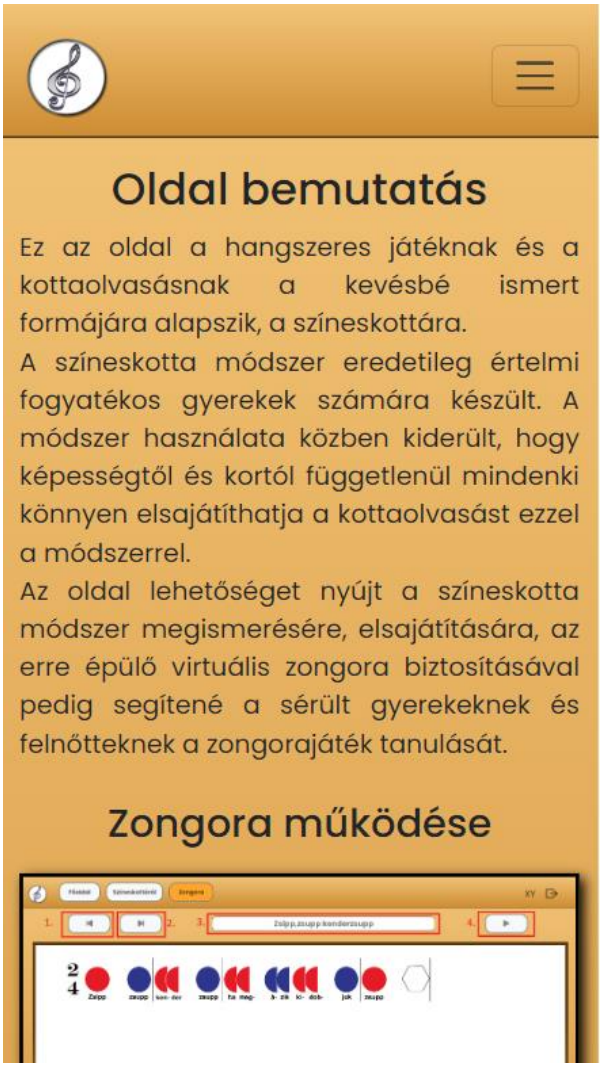
A zongora kottája felett több gomb is látható, amik az egyes funkciókért felelnek. A dal címének a bal oldalán található két gomb a dalok közötti léptetésért felelős, a jobb oldalán pedig, alapállapotban egyetlen lejátszást elindító gomb áll rendelkezésre. A lejátszás gomb a specifikációban megfogalmazott dal meghallgatásáért felel, megnyomásával lejátszhatja az adott dalt a felhasználó. Megnyomásakor a lejátszás gomb ikonja lecserélődik egy ismétlés ikonra, amivel jelzi, hogy az újbóli megnyomásával újra kezdheti a meghallgatást. Emellett megjelenik egy külön gomb a dal lejátszásának megállítására. Dal végeztével a gombok visszaállnak az alaphelyzetre, a lejátszás ikon visszatér az ismétlés ikon helyére, illetve a megállítás gomb eltűnik.

A regisztráció és az elfelejtett jelszó visszaállításának esetében a felhasználónak meg kell adnia a jelszavát. A biztonság érdekében a jelszó megfelelő formátumához több megkötést is tettem, amelyek súgó segítsége nélkül esetleg problémát okozhatnak a látogatóknak. Ebből kifolyólag a jelszó megadás mezők mellé egy súgót helyeztem el, amely kurzor rávitel estén megjeleníti, hogy miknek kell szerepelni a megadott jelszókban (11.ábra).



11. ábra: Súgó a megfelelő jelszó formátumhoz (saját szerkesztésű ábra)

A webalkalmazásom felületénél a terv szerint törekedtem a reszponzív megjelenítésre is. A reszponzív megjelenés elkészítéséhez a már említett *Bootstrap* által nyújtott segítségre hagyatkoztam. Az oldal telefonos kinézete a 12. ábrán található.



12. ábra: Az oldal mobil kinézete (saját szerkesztésű ábra)

5. Implementáció

A program felépítése a tervezésnél megfogalmazott MVC minta szerint készült, az oldalaknak külön vezérlőket és nézeteket készítettem. A nézetek esetében a közös elemeket egy `viewparts` nevű mappába helyeztem el, amiket onnan hívok meg az oldalak megjelenítésénél.

A mappa struktúrát kettő részre bontottam, publikusra és privátra. A publikus magába foglalja a felhasználói oldalhoz tartozó mappákat és fájlokat, ilyen például: a *CSS-nek, JS-nek,* a képeknek a mappája, illetve a fő `index.php`, amire minden kérés során navigálva lesznek a felhasználók. A privátba kerültek a további mappák és fájlok, amik a működésért felelnek például: az MVC modell részei és egyéb működéshez szükséges állományok. Emellett egy `.htaccess` fájllal letiltottam a felhasználók mappához való hozzáférését biztonsági okokból.

5.1. Útválasztó

Mielőtt kitérnék a specifikációban megfogalmazott funkciók implementálására, még fontosnak tartom kiemelni az oldalak közötti navigálásért felelős útválasztót (Router). A működéséhez készítettem egy `.htaccess` fájlt, ami megalapozza a navigálásnak a működését. A fájl feladata, hogy ha a kérés egy meglévő állományra esett, akkor letölti, ha pedig olyanra, ami nem létezik, akkor az `index.php` fájlra navigálja. Utóbbira elsősorban azért van szükség, mivel ahogyan azt említettem a privát mappában lévő fájlokhoz nem férhet hozzá a felhasználó, így az alkalmazás nem létező fájlként kezeli őket és az `index` állományra vezeti a látogatót. A hozzáférhetetlen oldalaknak az elérésére és megjelenítésére készítettem egy külön `Router` nevű osztályt, ami lehetővé teszi a további oldalak elérését. Az osztály három statikus metódusból áll. A fő metódus a `routingTo($fullUrl)`:

```
static function routingTo($fullUrl){
 $path = Router::getUrlPath($fullUrl);
 if(file_exists('../private/Control/'.$path.'.ctrl.php')){
   require('../private/Control/'.$path.'.ctrl.php');
 }
 else{
   $path == '/' ? require('../private/Control/index.ctrl.php') : header('Location:
/error?code=404');
 }
}
```

A metódus lehetővé teszi az oldalak és a hozzájuk tartozó vezérlők közötti koordinálást. A koordinálás kivitelezéséhez először az osztály egy másik statikus metódusát a `getUrlPath($url)`-t használva kinyeri a URL címbe írt utat és elmenti egy változóba. Ezt követően a *PHP*-be épített `file_exists()` függvénnyel megvizsgálja, hogy a URL-ben megadott útnak létezik-e vezérlője, ha igen akkor meghívja, ha nem akkor ellenőrzi, hogy üres-e. Üres út megadás esetében a főoldalnak a vezérlőjét (`index.ctrl.php`) hívja meg, azonban ha az előzőek egyike se igaz, akkor azt jelenti, hogy a megadott út nem létezik. Nem létező út előfordulására egy külön hiba vezérlőt készítettem, ami szövegesen jelzi a felhasználónak, hogy az általa kívánt oldal nem létezik.

5.2. Regisztráció

A regisztráció egy kötelező funkció, hiszen az alkalmazás úgy lett megtervezve, hogy hiányában nem lehet elérni a zongora szolgáltatást. A regisztrációhoz a fejlécen található, számára készített gombra kell kattintani, ami után a webalkalmazás a regisztrációs oldalra navigálja a felhasználót. Sikeres regisztrálás érdekében meg kell adni egy nem létező felhasználónevet, egy helyes formátumú email címet és kétszer a megfelelő erősségű jelszót. Későbbiekben a megadott adatok ellenőrzésére kerül sor, ha valami nem megfelelő vagy helytelen akkor az oldal hibaüzenetekkel jelzi a felhasználónak. Amennyiben nem ütközött hibába, akkor egy fiók megerősítő emailt küld a megadott email címre és a sikeres regisztrációról értesíti az adott személyt.

A folyamatot a regisztráció vezérlője, a `regisztracio.ctrl.php` fájl hajtja végre. Elsősorban a fájl ellenőrzi, hogy a felhasználó már be van-e jelentkezve. Az ellenőrzéshez egy bejelentkezett munkamenet változó értékének a meglétét vizsgálja. Ez egy fontos lépés, mivel kihagyásával a bejelentkezett személy a böngésző címsorát használva képes elérni a regisztrációs oldalt, ami további hibákhoz vezethet. Az alkalmazás ilyenkor megoldásként a főoldalra navigálja a felhasználót.

A vezérlőt úgy készítettem, hogy GET és POST metódussal érkezett kéréseket dolgozzon fel. Ennek okául az első ellenőrzést követően megvizsgálja, hogy milyen metódussal érkezett a webszerverhez. GET metódussal érkezett kérések jelentik számára, hogy adatok feldolgozása helyett, csak a felületet kell megjelenítenie. A felület egy regisztrációs űrlapot tartalmaz, ami az előző fejezetben már bemutatásra került. Az

űrlapba írt adatokat leadáskor POST metódussal visszaküldi saját magának, ekkor kezdődik meg az elküldött adatok ellenőrzése és feldolgozása.

A regisztrációkor megadott adatokat muszáj tárolni a működés során, ennek érdekében szükség van az adatbázis felhasználására. Adatbázis műveletekhez egy külön `Model` osztályt készítettem, ami négy privát tulajdonsággal rendelkezik. A tulajdonságok tartalmazzák a szükséges információkat az adatbankhoz való kapcsolódáshoz.

A kapcsolat létrehozása a konstruktorban valósul meg, hogy az osztály példányosítása során maga az összeköttetés is azonnal létrejöjjön. A privátok mellett még található egy védett `$conn` nevű tulajdonság, ami a PDO kapcsolatot tárolja. Továbbá a `Model` osztály metódusokat is tartalmaz, amik a létrehozás, lekérdezés, módosítás és törlés (CRUD) műveletekért felelnek. A metódusok készítésekor törekedtem arra, hogy minden esetben használhatók és rugalmasak legyenek.

A regisztrációnál és a bejelentkezésnél megadott adatok ellenőrzéséhez szintén egy külön osztályt készítettem, ami a `Model` leszármazottja és a `Validator` elnevezést kapta. Az osztály saját privát tulajdonságokkal rendelkezik, amik a bemeneti adatokat tárolják. Példányosításkor a tulajdonságokat a konstruktorban tölti fel a megfelelő adatokkal.

A `Validator` több ellenőrző metódust tartalmaz, amelyek a tervezéskor megfogalmazott ellenőrzési lépéseket végzik el. A metódusok a `Model` szülő osztály lekérdező metódusát, *PHP*-be beépített függvényeket és reguláris kifejezéseket használnak.

A bejövő adatok ellenőrzéséhez, tehát a program elsősorban példányosítja a `Validator` osztályt, aminek paraméterként át kell adni a POST szuper globális változókban lévő adatokat. Ezt követően a már említett metódusokat felhasználva ellenőrzi, hogy a szükséges adatok nem maradtak-e ki és megfelelnek-e a tervezésben meghatározottaknak. Hibák esetén a metódusok külön munkamenet változókat hoznak létre, amelyeknek értékül hibaüzenetet adnak, valamint hamis logikai értékkel fognak visszatérni. Amikor működése során nem fut hibába, igazzal tér vissza. Egy ellenőrzést végző metódusra példa a `checkUserName()` metódus, ami alább látható:

```
function checkUserName(){
        if(empty($this->username)){
            $_SESSION['unameError'] = "Ne hagyja üresen a mezőt!";
            return false;
        }
        else if($this->Read('username','users','where username =
?',[strtolower($this->username)])->fetch() !== false){
            $_SESSION['unameError'] = "Nem megfelelő felhasználónév!";
            return false;
        }
        return true;
    }
```

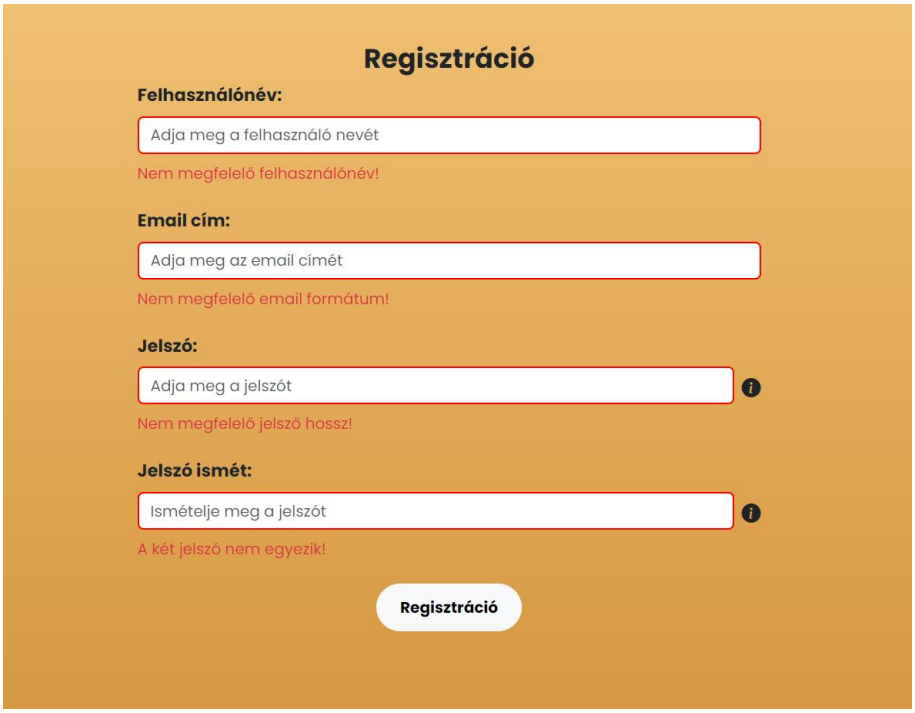
Abban az esetben, ha mindegyik metódus igazzal tér vissza, akkor először a fiók megerősítéséhez szükséges kulcsot generálja le és a biztonság érdekében kódolja is. A kulcsnak nehezen kitalálhatónak és biztonságosnak kell lennie, ezért a *PHP-be* épített `random_bytes()` függvénnyel 32 hosszúságú random bitet generáltam és hozzáfűztem a megadott felhasználónevet. Továbbá a `bin2hex()`-el hexadecimálisba konvertáltam át, majd pedig az `md5()` függvénnyel keletkezett hash értéket a `$vkey` változóba tároltam. A kulcs generálásáról szóló kódrészlet alább látható:

```
 $vkey=md5(bin2hex(random_bytes(32)).$_POST['username']);
```

Következőként a specifikációban megfogalmazott elvárásnak megfelelően kódoltam a megadott jelszót, `password_hash()` függvénnyel. A kódoló függvényen belül a `PASSWORD_BCRYPT`-et választottam.

A megerősítő kulcs és a kódolt jelszó mellett még szükség van a fiók aktiválás határidőjének a meghatározására és tárolására. Határidőnek a regisztráció idejétől számítva kettő napot adtam. Szükséges adatok létrehozása és változóba mentése után a `Model` osztály `Create()` metódusát használva a felhasználó regisztrálása megvalósul és bekerül az adatbázisba, a megerősítő email elküldésre kerül a megadott email címre. Ezek mellett egy sikeres munkamenet változót hoz létre igaz logikai értékkel, a sikeres regisztráció nézet megjelenítésének jelzésére.

Másik esetben, ha legalább egy hibába futott az ellenőrzés, akkor visszairányítja a felhasználót a regisztrációs oldalra és a hibát kiváltó adat mezője alatt pirossal jelzi a megfelelő hibaüzeneteket (13.ábra).



13. ábra: A regisztrációs felület hibák esetén (saját szerkesztésű ábra)

5.3. Bejelentkezés

A bejelentkezés működéséért a saját vezérlője felel. A funkció működése több dologban hasonlít a regisztrációhoz. A bejelentkezésnél is lehetetlenné kell tenni az oldal elérését abban az esetben, ha a felhasználó már be van jelentkezve, illetve a vezérlő a regisztrációéval megegyezően dolgozza fel a GET és POST metódussal érkezett kéréseket.

A POST kérés esetén a `Validator` pédányosítása következik. Az űrlapon megadott adatok megfelelősségének ellenőrzéséhez, az osztálynak egy külön metódusát használom, a `checkLoginInput()`-ot. A `checkLoginInput()` csak a bejelentkezési adatok vizsgálatára és a művelet elvégzésére készült. Legelőször egy `$isvalid` nevű változót deklaráltam igaz logikai értékkel. A változó értéke egyes vizsgálatok hibába ütközése során hamisra változik. Később ettől függ, hogy a vezérlő milyen felületet jelenít meg.

A változó deklarálása után következik a felhasználónév és jelszó meglétének ellenőrzésére `empty()` függvény segítségével. Amennyiben a kettő közül valamelyiknek a kitöltése kimaradt, akkor egy munkamenet változóba menti a

hibaüzenetet és az `$isvalid` változó értékét hamisra állítja. A munkamenetben tárolt hibaüzenetet kiírja a hibát okozó bemenet alá.

Feltéve, ha eddig nem ütközött hibába, akkor a `Model Read()` metódusával lekérdezi a megadott felhasználónévvel egyező rekordot az adatbázisból és ellenőrzi, hogy a benne található kódolt jelszó megegyezik-e a bejövővel. A jelszó ellenőrzéshez a *PHP* által nyújtott `password_verify()` függvényt használom, ami szemügyre veszi, hogy a megadott egyezik-e a kódolttal. Hogyha valamelyik esetén hiba lép fel, akkor a már használt módon értesíti a felhasználót arról, hogy hibás felhasználónevet vagy jelszót adott meg. Erről a kód alább tekinthető meg:

```
 if($isvalid){
            $result = $this->Read('*','users','where username =
?',[strtolower($this->username)])->fetch();
            if($result){
                if(password_verify($this->pwd, $result['password']) &&
$result['deleted'] == 0){
                    $_SESSION['loggedin'] = true;
                    $_SESSION["userId"] = $result["id"];
                    $_SESSION["userName"] = htmlspecialchars($result["username"]);
                    return $result;
                }
            }
            $_SESSION['loginUnameError'] = "A felhasználó név vagy jelszó hibás.";
            $_SESSION['loginPwdError'] = "A felhasználó név vagy jelszó hibás.";
        }
```

Hibamentes vizsgálat során a program elvégzi a bejelentkezési műveleteket, létrehozza és értékekkel látja el a különböző bejelentkezéssel kapcsolatos munkamenet változókat. A metódus lefutása után a program visszatér a bejelentkezés vezérlőhöz, ahol meghívja a megfelelő nézetet, ami vagy a főoldal bejelentkezett fejléccel, vagy a bejelentkezés oldal az előfordult hibaüzenetekkel megjelenítve.

5.4. Fiók megerősítés emailen keresztül

A regisztráció végén az alkalmazás egy fiók megerősítő emailt küld a felhasználónak az általa megadott email címre. A webalkalmazás több típusú emaileket küldhet ki működése során, emiatt úgy gondoltam megfelelő lenne egy `Mailer` osztály készítése, ami a küldéssel kapcsolatos folyamatokat tartalmazza. Az osztály három statikus metódussal rendelkezik:

1. `sendVMail()`: Ez a metódus felel a megerősítő email kiküldésért.

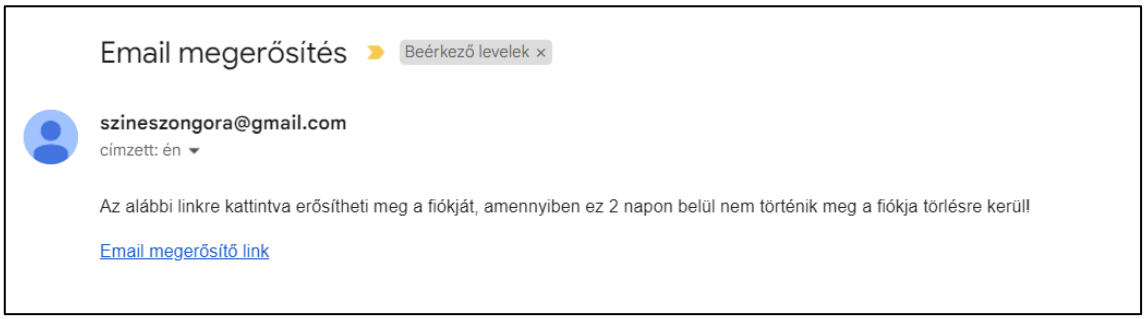
2. `sendRMail()`: A `sendRMail()` küldi az emailt, amin keresztül a felhasználó le tudja cserélni a jelszavát, abban az esetben, ha elfelejtette.
3. `sendDmail()`: A harmadik metódus feladata, hogy értesítő emailt küldjön az adott személynek a fiókjának törléséről, ha nem erősítette meg a határidőn belül (2 nap).

A `sendVMail()` kódja alább látható:

```
 static function sendVMail($email, $vkey){
        $subject= 'Email megerősítés';
        $message= "<p>Az alábbi linkre kattintva erősítheti meg a fiókját,
amennyiben ez 2 napon belül nem történik meg a fiókja törlésre kerül! </p>";
        $message.= "<a href='http://localhost/megerosites?vkey={$vkey}'>Email
megerősítő link </a>";
        $headers="MIME-Version: 1.0" . "\r\n";
        $headers.="Content-type:text/html;charset=UTF-8"."\r\n";
        $headers.="From: <szineszongora@gmail.com>"."\r\n";
        mail($email, $subject, $message, $headers);
    }
```

Az emailek küldéséhez a *PHP* beépített `mail()` függvényét használtam. A `sendVMail()` kettő paramétert kér be. Egyik az email cím, amire küldenie kell az emailt, a másik pedig a megerősítő kulcs, ami majd a fiók megerősítő linkbe kerül. A metódus még rendelkezik három változóval, amik a levél felépítéséért felelnek, ezáltal szükségesek és értékeiket paraméterekként át kell adni a `mail()` függvény működéséhez.

Az alkalmazás által elküldött email prezentálására az alábbi 14. ábra szolgál.



14. ábra: A megerősítő email (saját szerkesztésű ábra)

Az emailben található link a `megerosites.ctrl.php` vezérlő fájlt kéri le a webszerverről. A vezérlőnek célja szimplán, hogy megpróbálja elvégezni a megerősítést

és a sikerességének megfelelő üzenetet jelezni a felhasználó felé. A fájl kódja az alábbi részben látható:

```
if(isset($_GET['vkey'])){
    $vkey = $_GET['vkey'];
    $model = new Model();
    $result = $model->Read('verified, vkey','users','where vkey = ? and verified =
0 and deleted = 0 LIMIT 1 ',[$vkey])->fetch();
    if($result){
        $model->Update('users', 'verified=1', "where vkey = ?",[$vkey]);
        $valid = true;
        $title="Sikeres fiók megerősítés";
    }
    else{
        $valid = false;
        $title="Sikertelen fiók megerősítés";
    }
    require('../private/View/verify.view.php');
}
else{
    header('Location: /');
}
```

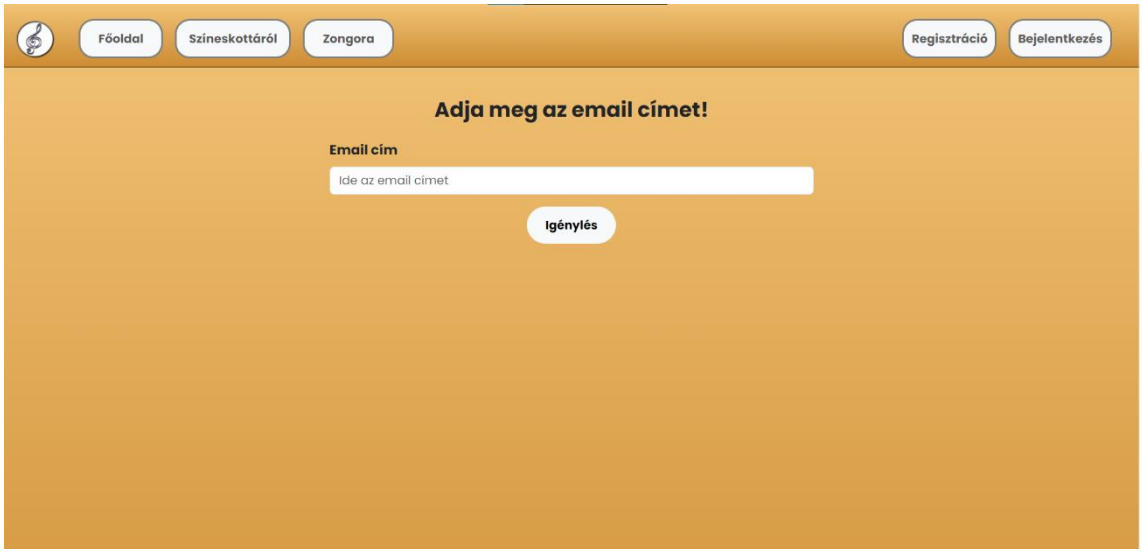
A szkript elsőként ellenőrzi, hogy a felhasználó a linken keresztül jutott-e el ide, ezt a `vkey` paraméterben való meglétével teszi. Ha a kulcs nem létezik akkor a fájl átirányítja az adott személyt a főoldalra.

Abban az esetben, ha a személy a linken keresztül jutott el a fájlhoz, akkor egy változóban lementi a paraméterben lévő megerősítő kulcsot. Következőkben példányosítja a `Model` osztályt és lekérdezi az adatbázisból a kulcsnak megfelelő felhasználó adatait. A lekérdezés eredményétől függően változtatja az oldalnak a címét és ad egy logikai értéket a `$valid` változónak, ami meghatározza, hogy milyen szöveg fogadja majd a látogatót. A sikertelen lekérdezés esetében egy hibaüzenetet jelenít meg. A sikeresnél pedig az eredményes megerősítésről tájékoztat, illetve a `Modelnek` az `Update()` metódusával átállítja a felhasználónak a `verified` mezőjét az adatbázisban. Az `Update()` metódus felépítése alább látható:

```
function Update($table,$setpart,$condition,$params=[]){
        $query = "UPDATE {$table} SET {$setpart} {$condition}";
        $stmt = $this->conn->prepare($query);
        $stmt->execute($params);
    }
```

5.5. Elfelejtette a jelszavát

Napjainkban mindegyik regisztrációval rendelkező weboldal ad lehetőséget a jelszavak visszaállítására akkor, ha a felhasználó elfelejtette és nem tud belépni. Az opció hozzájárul az oldal felhasználó barátiságának a növeléséhez. A funkciót kettő oldalra osztottam fel és mindkettőnek külön vezérlőt készítettem. Az első oldal, akkor jelenik meg, ha a felhasználó igénybe venné a szolgáltatást. Itt kell megadnia az email címet, amire várja a jelszó visszaállítással kapcsolatos emailt, ennek a felülete a 15. ábrán tekinthető meg:



15. ábra: A jelszó visszaállítás első oldala (saját szerkesztésű ábra)

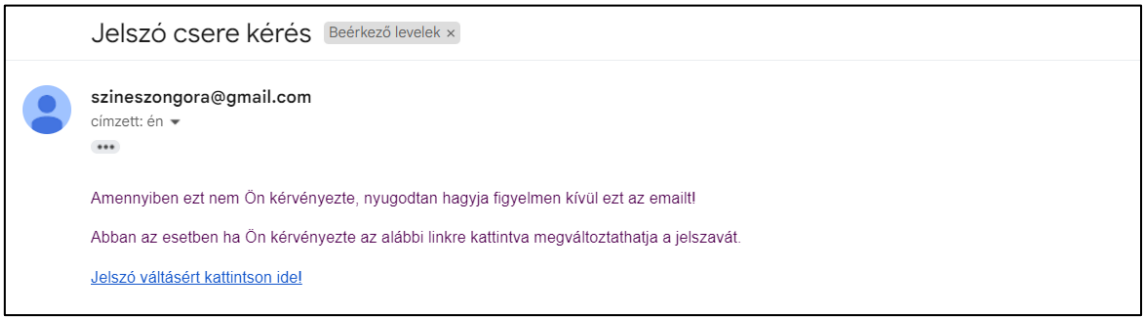
A felület megjelenítéséért és működéséért a `jelszokeres.ctrl.php` fájl felel. Akárcsak a regisztrációnál és a bejelentkezésnél, a két oldal felületét csak akkor szabad elérni, ha a felhasználó nincsen bejelentkezve, a vezérlők ezt ellenőrzik legelőször.

A jelszókérés vezérlője GET metódussal érkezett kérés esetén, elsősorban egy vizsgálatot végez azzal kapcsolatban, hogy a kérés egy már hibás jelszócsere után tért-e vissza. Ehhez a kérés metódust ellenőrzi és azt, hogy a URL-ben létezik-e az `error` paraméter. Mindkettő igaz esetén a `Router` osztály statikus `getUrlQuery($url)` metódusával kinyeri a URL paraméterét, amit kettő részre felbont és elment egy változóba. Erre azért volt szükség, mivel így a következőkben ellenőrizhetem a jött hiba paraméternek az értékét. Hibának megfelelő érték esetén a `$hibasEmail` változót igaz

értékkel látja el. Utána az oldal megjelenítése következik, ennél a résznél lesz fontos a változó, mivel igaz logikai értéke fogja jelenteni, a hibaüzenetek megjelenítését a bemenetek alatt.

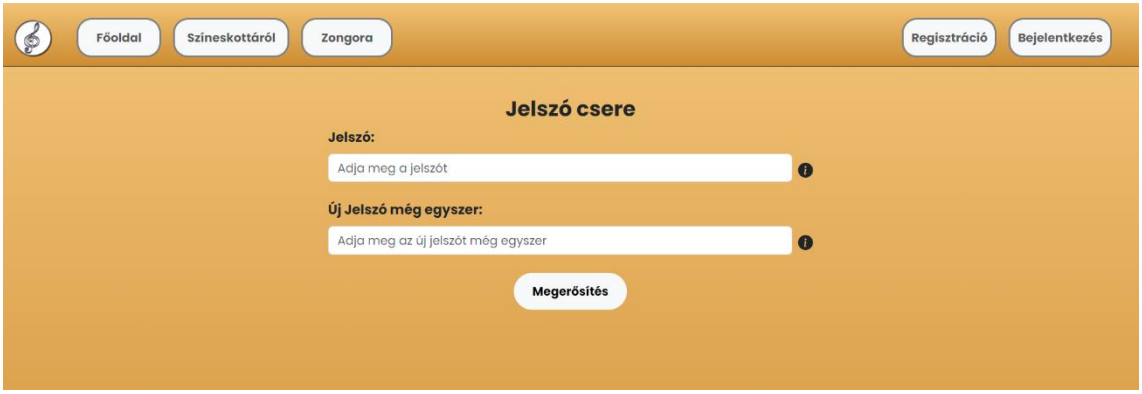
A GET metódus mellett ellenőrzi a POST kérést is, ami esetében fogadja a kitöltött űrlapot. Itt fog megtörténni a megadott email cím ellenőrzése, hiba keresése, a csere adatainak generálása, tárolása az adatbázis táblába és az email elküldése. Email küldéséhez a megadott cím mellett, generálni kell egy megerősítő tokent és le kell kérni a cserének az azonosítóját. A generált token és az azonosító az autentikációért felelnek. Továbbá egy dátumot is generálni kell, ami az email lejárati idejét jelenti.

Ezután az első szükséges ellenőrzés a megadott email címmel rendelkező felhasználó megléte. Hiba esetén frissíti az oldalt előbb már említett `error` paraméterrel és kiírja a hibaüzenetet. Abban az esetben, ha talált ilyen felhasználót, akkor először kitörli a `resetpassword` táblában már ezzel a címmel létező rekordokat, annak érdekében, hogy minden látogatóhoz csak egy rekord tartozzon és redundancia ne forduljon elő. Ezt követően új rekord jön létre, ami során a megerősítő token hashelt állapotban kerül tárolásra. Új rekord létrejöttekor az adatbázis egy azonosító értéket hoz létre a `ResetPwdID` mezőben, ami mindig egyel nagyobb szám, mint az előző bejegyzés, valamint az elküldendő emailben a felhasználó azonosítását segíti. Következő lépésként az azonosítót és a megerősítő tokent a levélbe téve, elküldésre kerül az email a megfelelő címre (16.ábra).



16. ábra: A jelszó csere kérés levél (saját szerkesztésű ábra)

A jelszó visszaállítás második oldala felel az új jelszó megadásáért. A felületen két bemeneti mező figyelhető meg a jelszónak és annak megismétléséhez (17.ábra).



17. ábra: A jelszó visszaállítás második oldala (saját szerkesztésű ábra)

A jelszócserének megfelelő üzemeléséért a `jelszovaltas.ctrl.php` vezérlő felel. A vezérlő az előzővel megegyezően dolgozza fel a GET és POST metódussal érkezett kéréseket. Az oldal eléréséhez a levélben megkapott linkre kell kattintani, emiatt először itt is GET metódussal fog megérkezni a kérés.

Fontos lépés elsőként az azonosító és a token meglétét ellenőrizni a link paramétereiben, illetve azt, hogy a kérés egy már hibás jelszócsere után érkezett-e. Az utóbbit egy `failed` munkamenet változó értéke jelenti. Ha az azonosító vagy token kimarad, vagy a token nem egyezik meg az adatbázisban lévővel, akkor olyan hiba áll fent, ami következtén egy hibaüzenetet kell megjeleníteni.

Abban az esetben, ha a látógató a linkre kattintva érkezett a szükséges adatok léteznek és megfelelőek a feltételeknek, ekkor az űrlap felületét küldi vissza a szerver. Az oldalon rejtett bemenet értékeként tárolásra kerül az azonosító és a token. Az említett értékek tárolása egy szükséges lépés, hiszen a későbbiekben az autentikáció miatt, még szükség van az adatokra a kérés feldolgozása során. Az űrlap leadása küldi POST metódussal a kérést. Ekkor kerül sor a megadott új jelszó feldolgozására és a helyes formátumának az ellenőrzésére, új jelszóként való beállítására.

Legelőször a POST szuper globális változóban meglévő két adat változókban való eltárolása történik meg, illetve a kettő megadott jelszó értékének megadásával példányosításra kerül a már előző funkcióknál használt `Validator` osztály. Használatával ellenőrzésre kerül a kettő jelszó, mindkettő ki lett-e töltve, a formátumuk megfelelő-e és megegyeznek-e. A szkript POST ágában három ellenőrzés van, hibába ütközésük során visszairányítják az oldalra a felhasználót és megfelelő formában megjelenítik a hibaüzenetet. Az első a `Validator` jelszó vizsgálatai által visszatért

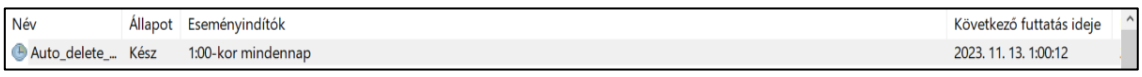
értékekre vonatkozik. Probléma fellépésekor a vezérlő frissíti az oldalt és a hibaüzeneteket megjeleníti az őket előokozó mezők alatt.

A jelszó hibátlan ellenőrzése következtében, a vezérlő egy változóba elmenti a jelenlegi dátumot, ami a további vizsgálathoz lesz szükséges. Következő a jelszó módosítás kérés érvényességére vonatkozik. Megvizsgálja, hogy a `resetpassword` táblában létezik-e a linkben lévő azonosítóval megegyező bejegyzés, illetve, hogy a kérés már lejárt-e. Hiba esetén értesíti a látogatót arról, hogy a kérése lejárt vagy érvénytelen, másik esetben a harmadik ellenőrzés következik, aminél a linkben lévő tokent kell összehasonlítani az adatbázisban lévővel. Ehhez a beérkezőt binárisba kell alakítani a hexadecimális formátumból majd a már használt `password_verify()` függvény segítségével össze kell egyeztetni az adatbázisban már meglévővel. Ha nem egyeznek akkor egy olyan hiba állt elő, ami után nem lehet újra megjeleníteni az űrlapot és a hiba oldalt kell prezentálni. Ellenkező esetben a fiók jelszavának a lecserélése történik meg. A jelszót kódolni kell, amihez a regisztrációnál használt módszert alkalmaztam, valamint a `users` adattáblában is frissíteni kell a jelszó mező értékét, majd üzenettel értesíteni kell a látogatót a sikeres jelszócseréről.

5.6. Automatikus fiók törlés és visszaállítási lehetőség

Ahogyan a specifikációkban meg lett határozva, az alkalmazásnak képesnek kell lennie töröltté tenni, azokat a felhasználókat, akik a megadott határidőn belül nem voltak képesek megerősíteni a fiókjukat. Ez azt vonja maga után, hogy akkor is meg kell történnie, ha a felhasználó éppen nem használja az oldalt.

A folyamat kivitelezéséhez egy Windows ütemezett feladatot hoztam létre, ami minden nap lefuttatja a törlést megvalósító szkriptet (18.ábra).



18. ábra: Az ütemezett feladat (saját szerkesztésű ábra)

A szkript a `scheduler.php` nevet kapta. Működése során először egy változóba menti az aznapi dátumot. Továbbá megvizsgálja, hogy az adott felhasználó fiókja már meg van-e erősítve, illetve, hogy a jelenlegi dátum nagyobb-e, mint a lejárati. Akikre az utóbbi igaz és még nem erősítették meg a fiókjukat, azok esetében szükséges az

adatbázisban igazra módosítani `deleted` mezőt. A funkció még elküld a felhasználó email címére egy levelet, amelyben értesíti a törlésről és annak okáról az adott személyt, valamint lehetőséget ad a fiók visszaállítására. Az emailről készült képet a 19. ábrán láthatja.

19. ábra: A fiók törlésről kapott email (saját szerkesztésű ábra)

A fiók visszaállítási lehetőség egy fontos és a felhasználóknak kedvező funkció, mivel előfordulhat, hogy a megerősítés technikai vagy életben történt váratlan események miatt maradt el. Abban az esetben, ha a felhasználó élne a visszaállítás lehetőségével, az emailben található link a fiók újraaktiválás vezérlőre irányítja a felhasználót. A link tartalmazza a visszaállítást kérelmezőnek az adatbázisban lévő megerősítő kulcsát. Meglétét ellenőrzi a vezérlő, ha valaki anélkül próbálná elérni, akkor átirányítja a főoldalra. Maga a visszaállítási folyamat a regisztrációnál alkalmazott működést használja újra, kisebb eltéréssel.

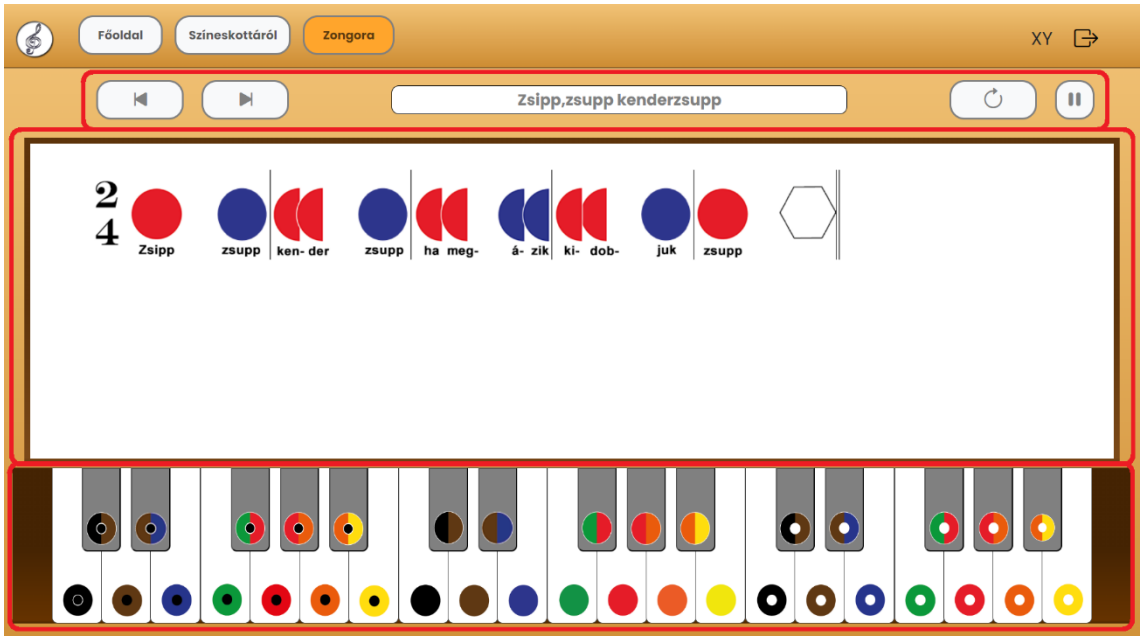
```
$result = $model->Read('*', "users", "where vkey = ?", [$key])->fetch();
    if($result){
        $vkey=md5(bin2hex(random_bytes(32)).$result['username']);
        $expiry_date = date("Y/m/d",time() + 172800);
        $model->Update("users", "deleted = 0, vkey = ?, expiry_date = ?", "where
username = ?", [$vkey, $expiry_date, $result['username']]);
        Mailer::sendVMail($result['email'],$vkey);
    }
    else{
        header('Location: /error?code=404');
        exit();
    }
```

Az eltérés abban, nyilvánul meg, hogy a jelszóval ebben az esetben nem kell műveletet végezni, valamint az új adatbázis rekord helyett a már meglévő, visszaállítást kért felhasználó rekordját kell módosítani. Az utóbbihoz a szkript megkeresi a linkben

található kulcsnak megfelelő felhasználó rekordját, ha megtalálta, akkor a már használt módon létrehoz egy új megerősítő kulcsot és egy új lejárati dátumot. Következőként módosítja a felhasználó rekordját, átváltoztatja töröltről még nem törölt állapotra, illetve lecseréli a kulcsot és a lejárati dátumot az újonnan generáltakra. A végeztével pedig visszavezeti a felhasználót a főoldalra.

5.7. Virtuális zongora

A virtuális zongora az alkalmazás témájából adódóan a legfontosabb funkció, megléte nélkül az alkalmazás nem tekinthető elkészültnek és az egész projekt értelmét veszti. A 20. ábrán megfigyelhető, hogy a zongora funkció három részre osztható: a zongora vezérlő sáv, a kotta és a zongorabillentyűk.



20. ábra: A virtuális zongora funkciói (saját szerkesztésű ábra)

A sávban található az éppen tanult dal címe, illetve a vezérlő gombok, amik felelnek a dalok közötti léptetésért, az adott dal meghallgatásáért, annak a megállításáért és újrajátszásáért. A dalok váltása AJAX kéréssel történik. A léptető gombok egy gombnyomás eseményre figyelnek, amelyek során a kliens egy GET metódus kéréssel fordul a webszer zongora vezérlőjéhez. A kérés cél URL címéhez hozzácsatoltam a jelenlegi dal azonosítóját, amely az oldal megjelenítésekor egy rejtett bemeneti mező értékeként került tárolásra. Erre a szerveroldali kérés feldolgozásakor van szükség.

A vezérlő elsősorban a specifikációban meghatározott feltételek megvalósulását vizsgálja, amiktől függően jeleníti meg a megfelelő tartalmat. Feltéve, ha a felhasználó jogosult a zongora használatára, a zongora megjelenítés mellett a `song` adatbázis táblából lekérdezi annak a dalnak az adatait, ahol a felhasználó éppen tart. A hangszer első megjelenítésekor az adatokat az oldalon található dinamikus elemek értékeinek állítja be, később még feldolgozza az AJAX kéréseket is. A feldolgozás működéséhez a URL címhez csatolt dal azonosítóra van szükség, ennek érdekében ellenőrzi a meglétét, majd a megkapott azonosítónak megfelelő dal adatait lekéri az adatbázisból. Siker esetén átállítja a felhasználó táblában az adott látogatónak a jelenlegi dal azonosító értékét és JSON formátumban szövegként visszaküldi válaszként. Amennyiben nem található a kért dal azonosító, akkor 404 hiba választ küld vissza.

A kliens a visszajövő válasz állapotát vizsgálja, hogy sikeres volt-e a vagy sem. Sikeres visszatéréskor a megkapott adatokat átalakítja JavaScript objektummá felhasználhatóságuk érdekében. Ezután a dinamikus tartalmak értékeit lecseréli és a további funkciókhoz szükséges értékeket visszaállítja eredeti állapotukra. Hiba válasz esetében egy hibaüzenetet jelenít meg a felhasználónak a sáv fölött, ami fél másodperc múlva elhalványul, majd eltűnik. Hiba akkor fordulhat elő, ha a léptetés során nulla vagy olyan dal azonosítóra lépett, ami túl halad az adatbázisban található utolsó.

A sávban található dal lejátszást irányító gombok közül kezdetben egy, a dal indítást eredményező gomb látható, aminek megnyomásával elindul az adott dal. A gombon látható ikon egy újrajátszást jelző ikonná változik és megjelenik a szünet gomb. A szünet lenyomására a dal megáll és lecseréli a rajta látható ikont egy indítást jelzőre. A gomb újbóli megnyomásával a dal onnan folytatódik, ahol megállította a felhasználó.

A vezérlő sáv alatt látható a jelenleg tanulni kívánt dal színes kottája. A megoldásához képeket használtam. A kotta alatt kerül megjelenítésre a zongora billentyűzete, ami gomb és hang elemekkel lett megoldva. Ezek `id` és `data-note` attribútumok segítségével lettek összekapcsolva. A billentyűzet egér kattintással működik, hiszen a fő célközönségbe tartozók számára nehezítené a használatot, ha a számítógépes billentyűzetük és a monitor között kellene változtatni a figyelmüket. Tehát ezzel egyszerűbbé téve az alkalmazást számukra. Mindegyik gomb elemhez egy kattintás esemény lett csatolva. Az esemény a `playNoteAudio()` függvényt hívja meg, ami a lenyomott hang `data-note` értékének megfelelő hang elembe ágyazott fájlt játssza le, így megszólaltatva a megfelelő zongora hangot.

6. Weboldalak védelme

Az internet megalakulása és elterjedése óta léteztek támadások a weboldalakkal szemben, azonban akkoriban mai szemmel nézve csak kisebb károkat okoztak. A technológia fejlődése számukra új támadási lehetőségeket és módszerek feltalálását eredményezte. Ennek eredményeképp a 21. században jelentősen megnövekedett a kibertámadások száma és azoknak a fajtái. Napjainkban már olyan szintet értek el, hogy az ellenük alkalmazott kiberbiztonság 2023 és a jövő egyik legfontosabb tendenciái közé emelkedett. [19]

6.1. OWASP

Az Open Worldwide Application Security Project röviden OWASP, egy nonprofit alapítvány, ami a szoftverek biztonságának a növelésével foglalkozik. Az alapítvány 2001. december 1-jén indult, célja, hogy a szervezeteket segítse a megbízható alkalmazások készítésében és üzemeltetésében. Általuk készített eszközök, fórumok és dokumentumok ingyenes felhasználhatóak, amelyek közé tartozik az OWASP top 10 listája is. [20]

OWASP TOP 10:

A lista elsősorban egy figyelemfelkeltő dokumentum, amely különböző biztonsági kockázatokat tartalmaz. Célja, hogy a benne említett és azokon kívüli kockázatokra hívja fel a figyelmet. Fontos megemlíteni, hogy a listát elsősorban nem egy alapnak tervezték, ebből kifolyólag csak egy minimumnak, kiindulópontnak tekinthető, ami egy jó kezdést biztosít az applikációk védelmében. A jelenlegi legújabb listának még mindig a 2021-es tekinthető, emiatt a továbbiakban az ebben szereplőkről fogok szót ejteni. [21]

6.1.1. Biztonsági kockázatok és ellenük alkalmazható védekezési lehetőségek

A következőkben a TOP 10 listában megtalálható biztonsági kockázatokról ejtek szót, mit is jelentenek az adott pontok, mik jellemzők rájuk, illetve mit tehetünk az elhárításuk érdekében.

1. Hibás hozzáférés-szabályozás:

A hozzáférés-szabályozás felelős a felhasználók csak jogosultságaikon belüli műveletek végrehajtásáért, ami azt jelenti, hogy egy felhasználó csak olyan műveleteket képes végrehajtani, amik a szerepkörének megengedettek. Ezek a fajta hibák általában

jogosulatlan információ megjelenítéshez, módosításhoz, törléshez és a látogató szerepkörén kívül eső üzleti logikai funkciók végrehajtásához vezetnek. [21]

A hozzáférés-szabályozás sérülékenységei közé tartoznak:

A legkisebb privilégium vagy alapvető tagadás elvének a megsértése, ahol a hozzáférést csak egy adott felhasználó csoportnak, szerepkörnek kellene szolgáltatni, de bárki hozzáférhet, valamint hozzáférés ellenőrzés megkerülése a URL paraméterek vagy a *HTML* oldal megváltoztatásával. Továbbá ide tartozik még más felhasználó fiókjának a megtekintése vagy szerkesztése, illetve a felhasználóként való műveletek végzése bejelentkezés nélkül vagy az adminisztrátor műveletek végzése szimpla felhasználóként. [21]

A hozzáférés-szabályozással kapcsolatos kockázatok ellen több védekezési lehetőség áll fenn, ezek közé tartoznak: a nyilvános erőforrásokon kívül minden más alapértelmezett tagadása, a webszerver könyvtárak listázásának letiltása, a biztonsági másolat fájlok gyökér könyvtáron kívüli elhelyezése, valamint az ide tartozó hibák naplózása és az állapot munkamenet változók érvénytelenítése kijelentkezés után. [21]

2. ***Kriptográfiai hibák:***

A kriptográfiai hibák alatt az adatok védelmi igényeinek nem megfelelő teljesítését értjük. A jelszavak, bankkártya adatok, személyes információk különösen nagy védelmet igényelnek. E típusú hibák ellen a következők, illetve további intézkedések végezhetőek el: az érzékeny adatok meghatározása, szükségtelen érzékeny adatok tárolásának elkerülése, minden szükséges érzékeny adat és jelszavak titkosítása erős, sózott hashing-funkciók használatával. [21]

3. ***Injekció:***

Az injekciós támadások során a rosszindulatú felhasználó káros adatokat használ az adatbázisban meglévő érzékeny adatok kinyerésére. Egy alkalmazás sebezhetőnek mondható injekciót tekintve, ha a felhasználó által megadott adatokat nem ellenőrzi vagy nem szűri. A leggyakoribb injektálások közé tartoznak az *SQL*, *NoSQL* parancsok, objektum-relációs leképezés (*ORM*) injektálás. A sebezhetőség felfedezésére a legjobb módszerek a forráskód és a bemeneti adatok vizsgálata. [21]

A támadás megakadályozása érdekében elvárás az adatok, parancsoktól és lekérdezésektől való elkülönítése [21]. Ez alatt azt értjük, hogy a bemeneti adatokat

később szolgáltatjuk a lekérdezésnek, paraméterekként, így elkerülve az ellenséges adatok káros műveletének lefutását. Ebből adódóan a lehetőségek közé sorolható még egy biztonságos *API* használata, amely elkerüli az értelmezőnek a használatát, paraméterezett interfészt biztosít vagy objektum-relációs eszközökre tér át [21].

4. ***Bizonytalan tervezés:***

A bizonytalan tervezés egy olyan probléma, amely több biztonsági kockázatot tartalmaz. Azonban fontos, hogy ez nem forrása a listában felsorolt további kockázatoknak, hiszen különbség van a nem biztonságos tervezés és végrehajtás között. A kettő közötti különbségek eltérő okokkal rendelkeznek, illetve az általuk létrejött hibák javításai is eltérnek egymástól. [21]

A bizonytalan tervezés elkerülése érdekében már a tervezési fázisban fontos biztonságosra tervezni az alkalmazást, ennek okán az előfordulható veszélyeket meg kell határozni és azok ellen tervezni. A tervezés érdekében biztonságos tervezési mintákat, fenyegetésmodellezést, és egyéb biztonsági terveket készíthetünk. Ezt követően a megfelelősségüknek az ellenőrzésére kell, hogy sor kerüljön, egység- és integrációs tesztek felhasználásával. [21]

5. ***Hibás biztonsági konfiguráció:***

A hibás biztonsági konfigurációkba több minden tartozik, elsősorban akkor beszélhetünk hibás konfigurációról, ha az alkalmazás bármely részén nem rendelkezik megfelelő védelemmel. További hibás konfigurációs esetek, ha szükségtelen funkciók vannak engedélyezve vagy telepítve, alapértelmezett fiókok továbbra is engedélyezve vannak, a hibakezelés túl bőbeszédű rendszerről szóló hibaüzeneteket jelenít meg a felhasználók számára. Ezeken túl a frissített rendszereknél a biztonsági opciók ki vannak kapcsolva vagy adatbázisok, alkalmazáskiszolgálók biztonsági beállításai nincsenek beállítva. [21]

A hibamentes konfiguráció érdekében megismételhető védelmi folyamatokat használunk, amelyek a védelem mellett még jóval gyorsabbá teszik a fejlesztést. A különböző környezeteket azonos módon kell konfigurálni, azonban a hitelesítő adatoknak különbözőnek kell lennie az egyes részeknél. Továbbá ajánlott a minimális platform, felesleges funkciók és komponensek nélkül, illetve a részekre bontott felépítés. [21]

6. Sebezhető és elavult komponensek:

A hatodik pont magába foglalja a komponensekből származtatható biztonsági réseket, amiknek a kiváltó okai lehetnek, ha a használt komponensek verziója ismeretlen számunkra, elavult vagy sebezhető szoftvereket használunk, valamint a frissített komponensek tesztelése elmarad. Az alkalmazható védelmi lépések lehetnek a felesleges funkciók, komponensek eltávolítása, kliens- és szerveroldali leltározása és a komponensek csak hivatalos forrásból való beszerzése. [21]

7. Azonosítási és hitelesítési hibák:

A hitelesítési támadások az egyik leggyakoribbaknak mondhatók, alkalmazásukkal a támadó célja egy ártatlan felhasználó fiókjának vagy személyes adatainak a megszerzése. A kivitelezésük abban az esetben lehetséges, ha az alkalmazás által használt hitelesítés nem alapos. Ilyen gyengeségek, ha az alkalmazás lehetővé teszi az automatikus, nyers erővel működő vagy más automatizált támadások használatát, valamint hiányzó vagy nem hatékony többfaktoros hitelesítést használ, munkamenet azonosítót vagy egyéb személyes adatot jelenít meg a URL címben. Továbbá a sikeres bejelentkezés során újra felhasználja ugyan azt a munkamenet azonosítót az új generálása helyett, illetve a kijelentkezés során helytelenül érvényteleníti a használt azonosítókat. A jelszavakkal kapcsolatos gyengeségeknek számít, a gyenge vagy alapértelmezett jelszavak kreálásának engedélyezése és szimpla szöveges, nem titkosított formátumban való tárolásuk. [21]

Az ilyen típusú támadások kivédésére megoldást jelent, ha hatékony több faktoros hitelesítést alkalmazunk, ezzel megakadályozva az automatikus támadásokat. Fontos a gyenge jelszavak elleni vizsgálatok, a sikertelen bejelentkezési kísérletek közötti idő késleltetése, illetve szerveroldali, biztonságos munkamenet kezelő használata, amely a sikeres bejelentkezés során újra generálja a munkamenet azonosítót. [21]

8. Szoftver- és adatintegritási hibák:

A szoftver- és adatintegritási hibák alatt, azokat a kódokat értjük, amelyek nem védenek az integritás megsértése ellen. Manapság már sok alkalmazás automatikus frissítés funkciót használ, aminek esetében a frissítés a megfelelő integráció ellenőrzése nélkül történik. Ezt a jellemzőt használhatják ki a támadók, feltölthetik a saját frissítéseiket, amiknek célja a károkozók terjesztése. [21]

Az integritási problémák ellen érdemes a digitális aláírások használata, illetve annak ellenőrzésére, hogy a szoftver a megfelelő, nem változtatott forrásból származik. Továbbá fontos, hogy a kódot és a konfigurációkat csak felülvizsgálat elvégeztével lehessen változtatni, annak érdekében, hogy csökkentsük az említettekbe való kártékony kód kerülésének lehetőségét. [21]

9. Biztonsági naplózás és felügyeleti hibák:

A naplózás és megfigyelés a támadások, illetve helytelen műveletek észlelésében játszik fontos szerepet, nélküle a jogsértések észlelése jóval nehezebbé, szinte észrevehetetlenné válna. A hibák ilyesfajta kategóriájába olyanok tartoznak, mint például: az ellenőrizhető események naplózásának kihagyása, a hibák nem helytálló megfogalmazású naplózása, naplók csak helybeli tárolása és a riasztások nem valós vagy ahhoz közeli időben történése. [21]

A hibák elleni megoldások közé tartozik, ha biztosításra kerül az érvényesítési hibák naplózása, megfelelő felhasználói kontextussal és lejárati idővel, annak érdekében, hogy a támadó fiókok azonosíthatók legyenek és elegendő idő legyen a törvényszéki elemzéshez. Emellett biztosítani kell, hogy a napló fájlok a naplókezelők használatának megfelelő formátumában készüljenek és kódolva legyenek az esetleges injekciós vagy egyéb támadások elhárítása érdekében. [21]

10. Szerveroldali kéréshamisítás (SSRF):

Az SSRF hibák abban az esetben fordulhatnak elő, ha a webalkalmazás a megadott URL cím ellenőrzése nélkül kéri le webszerveren található erőforrást, így a támadó számára lehetővé válik, hogy az alkalmazás az általa szerkesztett kéréseket küldje a nem várt célállomásra. [21]

Az SSRF ellen védelmet nyújt az OSI modell hálózati rétegéből a távoli erőforrás hozzáférés funkciók szétválasztása külön hálózatokba, illetve az alapértelmezett tiltással kapcsolatos tűzfalszabályok vagy a hálózati hozzáférést meghatározó szabályok érvényesítése. A hálózati rétegben lévők mellett, az alkalmazási rétegben a felhasználók által megadott bemeneti adatok hitelesítése, a HTTP átirányítások tiltása és a felhasználóknak küldött nyers üzenetek elkerülése is védelmet biztosít. Továbbá az alkalmazási rétegben és azon kívüli további intézkedések is lehetségesek az SSRF megakadályozására. [21]

6.1.2. A saját weboldalba néhány implementált megoldás

Az alkalmazásom tervezése és fejlesztése során törekedtem a fentebb felsorolt pontok figyelembevételére, hogy egy minimális védelemmel rendelkezzen. Azonban a szűk rendelkezésre álló idő miatt, az elkészült szoftver még rendelkezhet biztonsági résekkel.

Első lépésként, a weboldal könyvtár struktúráját az implementáció fejezetben már ismertetett módon valósítottam meg, amivel lehetővé tettem, hogy a felhasználók csak a nyilvános erőforrásokat érhessék el. Figyelmet fordítottam annak, hogy a felhasználók csak a szerepkörüknek megfelelő funkciókat és oldalakat érhessék el, aminek eredményeképp a nem hitelesített fiókok és nem bejelentkezett látogatók nem használhatják a zongora funkciót, valamint a bejelentkezettek nem érhetik el a regisztráció, bejelentkezés és jelszócsere felületeket.

Következő lépésként a felhasználók azonosítására és adataiknak a védelmére implementáltam néhány megoldást. Kriptográfiai szempontnak megfelelően a szenzitív adatokat hashelt formátumban mentettem az adatbázisba. A kódoláshoz `Bcrypt` algoritmust használtam, amely mellé tizenkettes értéket adtam a cost paraméternek. A jelszavak kódolásán kívül még tettem annak érdekében, hogy csak erős jelszavak legyenek elfogadottak (minimum 8 karakter, legalább egy nagy betű, egy szám és egy speciális karakter), személyes adatok ne jelenjenek meg a URL sávban, kijelentkezéskor érvénytelenítse a munkamenet azonosítókat, valamint bejelentkezéskor cserélje le. Az utóbbihoz egy külön fájlt készítettem, a `SessionConfig.php`-t, amelynek a lényege, hogy az azonosítót 5 percenként újra generálja.

Az adatbázisban tárolt adatok védelmének érdekében a `Model` osztály adatbázis műveleteiért felelős metódusokat előkészített utasításokkal oldottam meg. Az előkészített utasítások esetében a bemeneti adatokat paraméterekként adjuk meg, ezáltal nem *SQL* kódként fogja feldolgozni és nem fog lefutni a kártékony bemenet.

További védelmi lépések az alkalmazásomban, hogy a *PHP* `.ini` fájljában kikapcsoltam a hiba üzenetek megjelenítését és saját magam által készített hibákat jelenítek meg, amelyek nem szolgáltatnak bő leírást az alkalmazásról a felhasználóknak. A bemeneti adatok kiírása során a *PHP* `htmlspecialcharts()` függvényt használtam, ami a *HTML* speciális karaktereket szöveggé alakítja, így védelmet nyújtva az *XSS* támadásokkal szemben.

Az *XSS* támadáson kívül, *Cross-site Request Forgery* támadások megnehezítése érdekében is próbáltam lépéseket tenni. Az alkalmazás összes űrlapjához egy véletlenszerűen generált CSRF tokent csatoltam rejtett bemenetként, amit továbbá munkamenet változóba mentettem, a későbbi felhasználása érdekében. Az űrlapok feldolgozásáért felelős műveletek előtt a token meglétét és a mentett változatával való megegyezését vizsgálom, amennyiben a kettő feltétel közül valamelyiknek nem tesz eleget, akkor a feldolgozás művelet nem valósul meg és visszairányítja a felhasználót a főoldalra.

7. Tesztelés és továbbfejlesztési lehetőségek

A fejezetben főként az általános tesztelési technológiákra, szintekre, valamint az alkalmazásom tesztelésére és annak eredményére térek ki. Továbbá szót ejtek a fejlesztésem során felmerült potenciális tovább fejlesztési ötleteimről is.

7.1. Tesztelés

Az alkalmazás implementációjának befejezése után következik az utolsó fejlesztési lépés, a tesztelés. A tesztelés során dől el, hogy az elkészült alkalmazás ténylegesen elkészültnek minősíthető-e, illetve rendelkezik-e még hibákkal.

A tesztelésre több technika létezik: fekete- és fehérdobozos. A feketedobozos tesztelés specifikáció alapú, ami azt jelenti, hogy a tesztelés során nem látjuk a forráskódot és a specifikációk alapján kell elkészítenünk a teszteseteket. A fehérdobozos esetében a forráskódra alapozzuk a tesztelést és mindig egy már kész struktúrát tesztelünk. [22]

A webalkalmazások tesztelése több szintre bontható, ezek a szintek: a komponensteszt, integrációs teszt, rendszerteszt és az átvételi teszt. A komponensteszt során a rendszer komponenseit külön-külön teszteljük, általában a forráskód tudatában. Továbbá a komponensek tesztelése között még megkülönböztetünk unit- és modultesztet. A unit-teszt a metódusokat teszteli, míg a modulteszt az alkalmazás nem funkcionális tulajdonságait vizsgálja, ilyen a sebesség. Az integrációs teszt a külön fejlesztett részek együttes működését teszteli, míg a rendszerteszt magának az egész rendszernek a működését. Átvételi teszt pedig, megegyezik a rendszerteszttel, annyi különbséggel, hogy ezt már nem a fejlesztők végzik, hanem a felhasználók. [22]

Fontosnak tartom megemlíteni, hogy a tesztelést általában több ember végzi el és erre számukra hosszabb idő áll rendelkezésre. Az én esetemben viszont egyedül végeztem el és a rendelkezésre álló idő miatt a teszteléseim során, inkább az alkalmazás főbb szolgáltatásainak és a helyes tartalom megjelenítésének a tesztelésére fektettem a hangsúlyt. Ebből adódóan fennállhat a valószínűsége, hogy kisebb, váratlan hibák fordulhatnak elő.

A tesztelésem során elsősorban fehérdobozos tesztelést használtam a komponens tesztek esetében, majd később a rendszertesztelésnél feketedobozos ellenőrzést végeztem. Első lépésben külön-külön teszteltem az alkalmazás által nyújtott funkciók részeit és az

alkalmazásba implementált védelmi megoldásokat. A funkciók esetében a megfelelő működés mellett, különösen figyelnem kellett a hibakezelésekre, a megfelelő hibaüzenetek nyújtására a felhasználók felé. Az említett tesztesetek jó eredményeket adtak vissza, ezáltal tovább léphettem a következő lépésre. Az integrációs tesztben a funkciók részeinek együttes működését vizsgáltam, ami után a rendszernek a tesztelése következett, ahol az egész webalkalmazás működését ellenőriztem. Ebben az esetben azt vizsgáltam, hogy az alkalmazás teljes egészében működik-e és a szolgáltatások megfelelnek-e a specifikációban megfogalmazott meghatározásoknak. A tesztekre pozitív eredményeket kaptam, a szolgáltatások megfelelően működnek és az alkalmazás is megfelelően használhatónak tekinthető.

7.2. Továbbfejlesztési lehetőségek

A fejlesztésem közben újabb és újabb ötletek merültek fel bennem, amik arra ébresztettek rá, hogy az alkalmazás jelenlegi verziójának lehetőségei, hasznossága és egy szinten való megfelelőssége ellenére még mindig messze van a legmagasabb potenciáljától. Az oldal következő szintre emelése érdekében továbbfejlesztési lehetőségekkel álltam elő, amelyeknek a megvalósítása és azokon túl további fejlesztések kigondolása a jövőbeli terveim közé tartozik. A következő tovább fejlesztési ötletekkel álltam elő:

- Az oldal további kottákkal való kiegészítése. A további kották meglétét szükségesnek érzem, mivel a weboldal jelenleg nem tartalmazza a tényleges tanításhoz szükséges mennyiséget.
- A webalkalmazások használata során előfordul, hogy a felhasználók önszántukból megváltoztatnák a fiókjuk adatait. Az alkalmazásom jelenlegi verziója erre nem ad lehetőséget, emiatt jövőbeli terveim között van egy fiók beállítás panel készítése. A panel az említett problémára szolgál megoldásként.
- A virtuális zongora oldal egy új gombbal való kiegészítése, amely megnyomásával a felület a színeskotta és a hagyományos kinézet között változtatható. Ezzel kezdeményeznék a normál képességi szintű felhasználók számára, akik inkább a hagyományos zongorát használnák.
- A virtuális zongora működésének egy mutatóval való bővítése, ami a dal játszása során mindig arra a hangra mutat, amelyiknél éppen tart a látogató. A

mutató hozzáadásának célja a felhasználók további segítése a dalok tanulásában, így növelve a felhasználói élményt és elégedettséget.

- Az alkalmazás védelmének bővítése, tovább növelése a jelenlegi megoldásokon túl, hogy tényleg biztonságosnak tudható legyen a weboldal.

Az elkészült webalkalmazásom jelenlegi verzióját, mindenesetre publikáltam. A publikáláshoz a nethely.hu [23] által biztosított szolgáltatásokat vettem igénybe. A publikált változatát kiegészítettem egy űrlappal, amit annak érdekében készíttettem, hogy jelenleg a még védelmi szempontból nézve nem teljes webalkalmazást csak egy zártkörű csoport érje el.

8. Összefoglalás

A szakdolgozatom során egy általam kitalált és elkészített webalkalmazás fejlesztését mutattam be. A motivációim közé tartozott, hogy a fejlesztett alkalmazásom hasznos és segítséget nyújtó legyen a társadalom számára. A célja a gyógypedagógusok munkájának, azon belül enyhén fogyatékos és tanulási zavarral rendelkező gyermekek fejlesztésének segítése. Ehhez a zongoratanítást helyezi a középpontba, azon belül a speciális színeskotta-módszer használatát. A webalkalmazás egy leírást tartalmaz a színeskottáról, illetve egy virtuális zongorát és kottákat, amik a módszernek megfelelően készültek.

A dolgozatomban kitértem a fejlesztés folyamán felhasznált technológiákra, az alkalmazás tervezésére, a grafikai felület és a főbb funkciók implementálására, illetve a tesztelési fázisra. Továbbá még szót ejtettem a webalkalmazások védelméről és a saját szoftveremben használt megoldásokról.

A tervezéskor megfogalmazott elvárások közé tartozott, hogy a weboldal olyan felülettel rendelkezzen, ami egyszerű, jól átlátható és a használt színekkel képes legyen élénkebbé tenni a megjelenítést. Ezenkívül a működéssel kapcsolatos elvárásokra vonatkozott, hogy valamilyen szintű védelmet, a mai weboldalaknál alkalmazott általános funkciókat és egy színeskotta-módszerre épülő virtuális zongorát biztosítson.

Végeredményül az elvárásoknak megfelelően elkészítettem a weboldalt. A jelenlegi verziója pedagógusok szempontjából könnyen és egyértelműen használható, míg a gyerekekéből figyelemelterelő elemek nélküli és tetszetős felülettel áll rendelkezésre. Emellett elérhetőek a mai weboldalak általános funkciói ilyen a regisztráció, bejelentkezés, elfelejtett jelszó cserélése. Továbbá egy alapszintű védelemmel rendelkezik és biztosítja a színeskotta-módszeren alapuló virtuális zongorát.

Az elkészített alkalmazásom a virtuális zongora meglétével képes a gyógypedagógusok munkáját segíteni. A munkájuk segítése több mindenben nyilvánul meg. Manapság már minden iskola és háztartás rendelkezik számítógéppel és internettel, így a pedagógusok a weboldal alkalmazásával akár olyan esetekben is felhasználhatják a zongorát a fejlesztéseik során, amikor egy valódihoz közvetlenül nem férnek hozzá.

Irodalomjegyzék

[1] Szabó Bálint, Webprogramozás I.,
Eszterházy Károly Főiskola nyomdájában, Egerben: dr. Kis-Tóth Lajos, 2013.

[2] Adonyiné Gábori Mária ,
„A sajátos nevelési igényű gyermek integrált/inklúzív nevelése,”
Pécs, PTE BTK Neveléstudományi Intézet, 2006.

[3] Tóth László, Sajátos nevelési igényű tanulók fejlesztése, Debrecen:
Debreceni Egyetemi Kiadó, 2015.

[4] „Nemzeti Köznevelési Portál,” [Online]. Available:
https://www.nkp.hu/tankonyv/enek_zene_6_ksni/lecke_06_001.
[Hozzáférés dátuma: 18 09 2023].

[5] Naár János, Weblap-szerkesztés lépésről lépésre, Debrecen:
Debreceni Egyetem Informatikai Kar Könyvtár-informatikai Tanszék, 2008.

[6] „Szabványkövető statikus honlapok szerkesztése HTML5 + CSS3 + SVG2”.

[7] Munk Sándor, „Szemantika az informatikában,” 2014..

[8] „WHATWG community,” 15 november 2023. [Online]. Available:
https://html.spec.whatwg.org/multipage/introduction.html#html-vs-xhtml.
[Hozzáférés dátuma: 20 09 2023].

[9] Nagy Gusztáv, Web programozás alapismeretek, Budapest:
Ad Librum Kiadó, 2011.

[10] „Bootstrap,” [Online]. Available:
https://getbootstrap.com/docs/5.3/getting-started/introduction/.
[Hozzáférés dátuma: 22 09 2023].

[11] Király Roland, Dinamikus weboldalak fejlesztése, Eger: Eszterházy Károly Főiskola Matematikai és Informatikai Intézet, 2010.

[12] „PHP Hivatalos Oldala,” [Online]. Available: https://www.php.net/manual/en/intro-whatis.php. [Hozzáférés dátuma: 28 09 2023].

[13] „PHP Hivatalos Oldala,” [Online]. Available: https://www.php.net/manual/en/intro-whatcando.php. [Hozzáférés dátuma: 28 09 2023].

[14] Szabó Bálint, Adatbázis fejlesztés és üzemeltetés I., Eszterházy Károly Főiskola nyomdájában, Egerben: dr. Kis-Tóth Lajos, 2013.

[15] „XAMPP Hivatalos Oldala,” [Online]. Available: https://www.apachefriends.org/hu/index.html. [Hozzáférés dátuma: 29 09 2023].

[16] „XAMPP Hivatalos Oldala,” [Online]. Available: https://www.apachefriends.org/faq_windows.html. [Hozzáférés dátuma: 29 09 2023].

[17] „Microsoft Support Oldala,” [Online]. Available: https://support.microsoft.com/hu-hu/topic/az-adatb%C3%A1zisok-tervez%C3%A9s%C3%A9nek-alapjai-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5#bmdesignprocess. [Hozzáférés dátuma: 03 10 2023].

[18] Dragos-Paul Pop és Adam Altar, „Designing an MVC Model for Rapid Web Application Development,” Elsevier Ltd., Romanian-American University, 1b Expozitiei Blvd., Bucharest, 012101, Romania, 2014.

[19] Bencze Áron, „Innotéka,” 5. augusztus 2019. [Online]. Available: https://www.innoteka.hu/cikk/fizikai_eroszak_nelkul.1952.html. [Hozzáférés dátuma: 14 10 2023].

[20] „OWASP,” [Online]. Available: https://owasp.org/about/. [Hozzáférés dátuma: 13 10 2023].

[21] „OWASP,” 2021. [Online]. Available: https://owasp.org/Top10/. [Hozzáférés dátuma: 13 10 2023].

[22] Ficsor Lajos, Dr. Kovács László, Krizsán Zoltán és Dr. Kusper Gábor, „Szoftvertesztelés,” Kelet-Magyarországi Informatika Tananyag Tárház.

[23] „Nethely.hu,” [Online]. Available: https://www.nethely.hu/. [Hozzáférés dátuma: 01 11 2023].

Ábrajegyzék

Táblázatjegyzék

Mellékletek jegyzéke

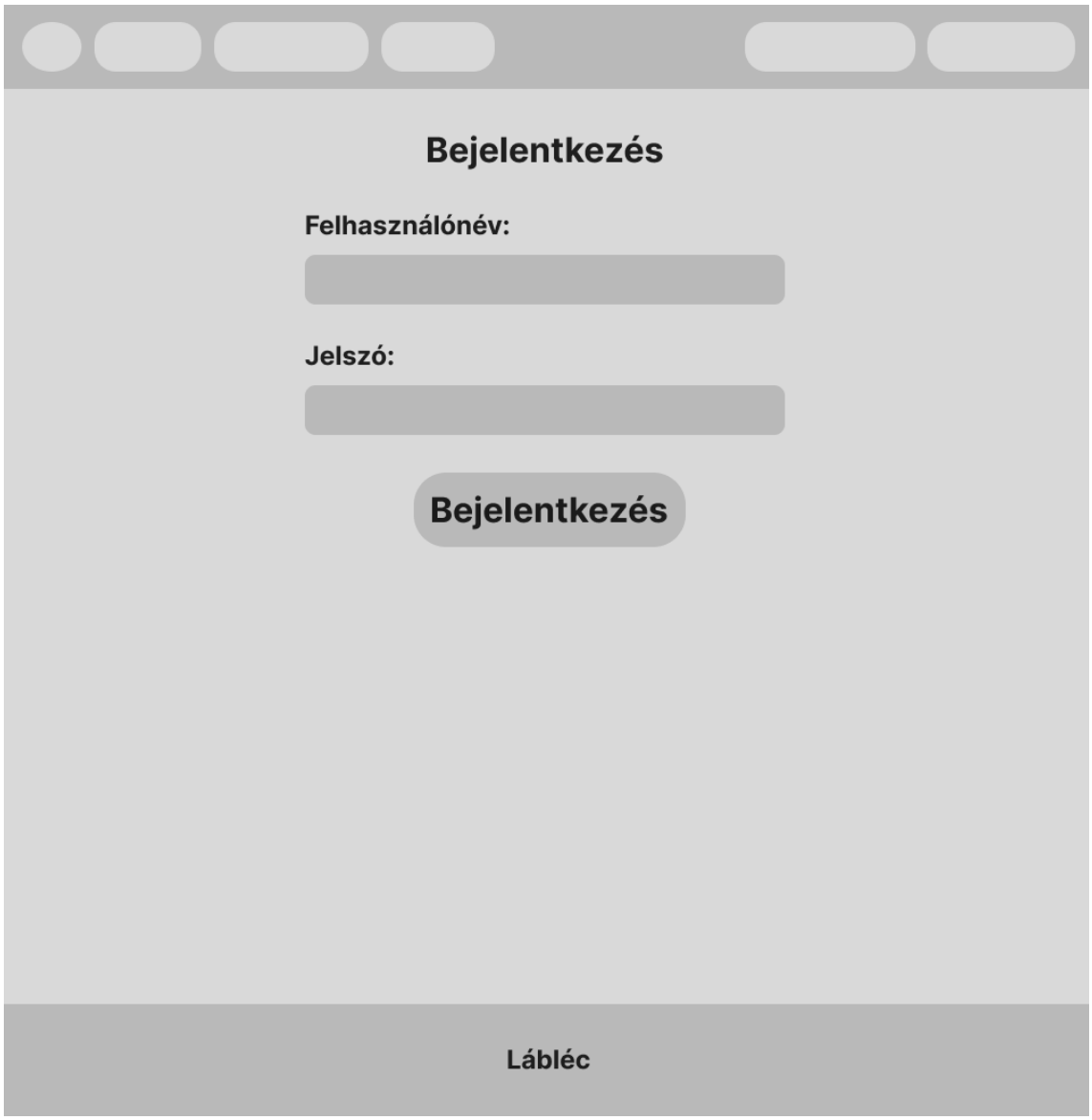
1. melléklet:

Grafikus felület további vázlatai

21. ábra: A színeskottáról oldal vázlata (saját szerkesztésű ábra)

22. ábra: A virtuális zongora oldalnak a vázlata (saját szerkesztésű ábra)

23. ábra: A regisztráció oldal vázlata (saját szerkesztésű ábra)

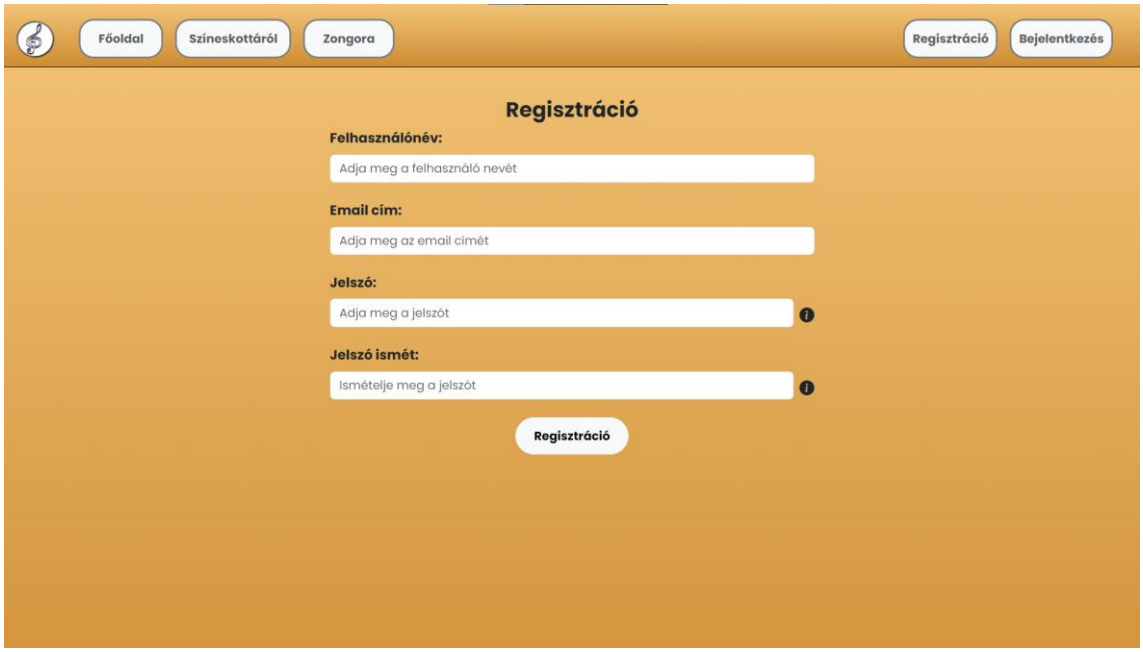


24. ábra: A bejelentkezés oldal vázlata (saját szerkesztésű ábra)

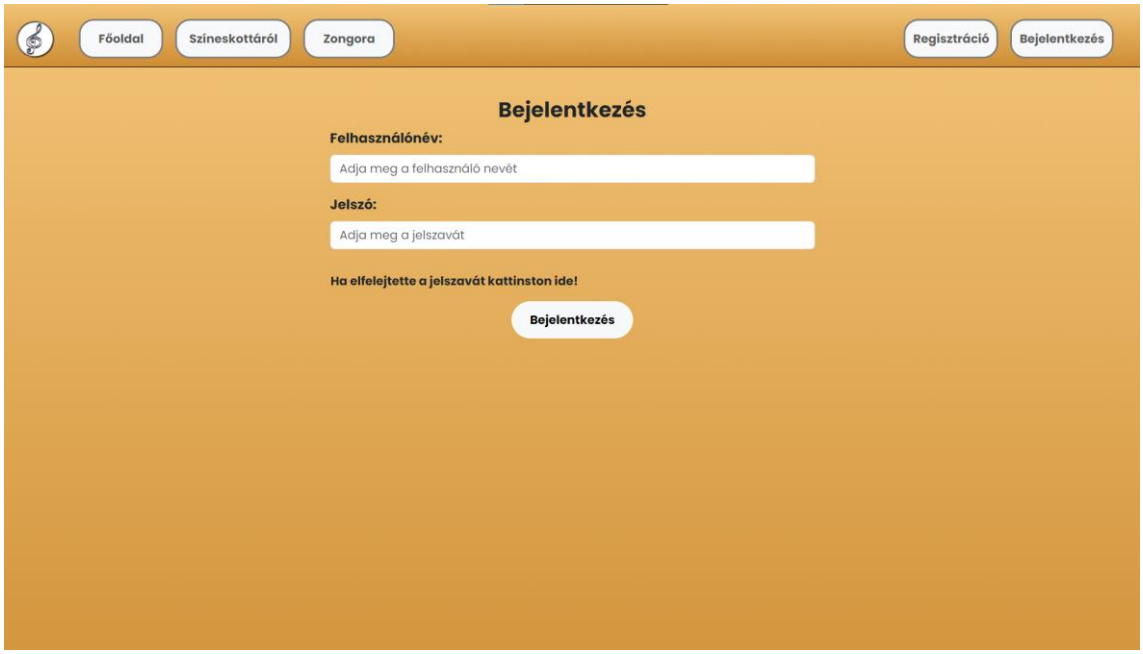
2. melléklet:

Elkészült grafikus felület további ábrái

25. ábra: Az elkészült színeskotta oldal (saját szerkesztésű ábra)



26. ábra: Az elkészült regisztrációs oldal (saját szerkesztésű ábra)



27. ábra: Az elkészült bejelentkezés oldal (saját szerkesztésű ábra)