XXI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2018. május 24–27.

Taboo

Akkordmenet szerkesztő és kezelő webalkalmazás

Szerzők:

Erőss Ervin

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Péter Adorján - Ferenc

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

drd. Sulyok Csaba, doktorandus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Kivonat

A Taboo projekt célja egy felhasználóbarát felület biztosítása akkordmenetek létrehozására és kezelésére. Célközönsége zenészek, akik böngészhetik és általuk létrehozott akkordmenetekkel bővíthetik az oldal tartalmát.

Az alkalmazás lényeges újítása az interaktív felület, amely lehetőséget biztosít akkordmenetek létrehozására és szerkesztésére. Egy felhasználó egyszerű egérműveletek segítségével könnyedén hozzáadhat akkordmeneteket, illetve törölheti vagy módosíthatja ezeket. Az akkordmenet megjelenítése és szerkesztése az alkalmazásban definiált DSL (domain specific language) felhasználásával történik. Ez a dinamikus kirajzolás garantálja az átláthatóságot különböző méretű eszközökön.

A dolgozat tartalmazza a projekt megvalósításához használt technológiákat, bemutatja ezen webalkalmazás kliens-szerver architektúráját, valamint rövid leírást tartalmaz az alkalmazás használatáról és továbbfejlesztési lehetőségeiről.

Tartalomjegyzék

Bevezető

Az internetet böngészve számos olyan weboldalra bukkanhatunk, amelyek akkordmenetek szerkesztésére adnak lehetőséget. A Taboo projekt motivációját elsősorban a már meglévő, ehhez hasonló weboldalak hiányosságainak a javítása jelentette. Célja, hogy felhasználóinak egy interaktív felületet biztosítson az akkordmenetek létrehozására és reszponzív design-t a megjelenítéshez. Az utóbbi által az alkalmazás megfelelő felhasználói élményt nyújt például tábortüzes zenéléseknél is, amikor csak mobiltelefon vagy tablet áll rendelkezésre. További hasznos funkcionalitások is fő szempontot képeztek tervezéskor, mint például az akkordmenetek PDF formátumban való letöltése, hogy azok offline módban is megtekinthetőek legyenek. A specifikáció részét képezte az akkordmenetek transzponálási lehetősége, vagyis ha az előadó más hangnemben szeretné játszani a dalt, ezt pár kattintással megvalósíthassa. További hasznos funkcionalitások is fő szempontot képeztek tervezéskor, mint például az akkordmenetek gyűjteményekbe való rendezésének a lehetősége.

A fent említett megvalósítások pótolják a hasonló weboldalak egyes hiányosságait, mint például az elfelejtett jelszó esetén nincs lehetőség új jelszó megadására, az akkordmenet lementése után nincs lehetőség utólagos módosításra, pontosításra. Ezek közül talán a legzavaróbb, ha új akkormenetet szeretnénk létrehozni. Mivel ezt csak szövegként vihetjük be a rendszerbe, ezáltal szinte lehetetlenné válik a pontos akkordmenetek létrehozása. Az akkordokat csak szóközök segítségével lehet a dalszöveg felé pozicionálni. Kisebb képernyővel rendelkező készülékeken az így bevitt szövegek sok esetben olvashatatlanná válnak a helytelen sortörés miatt. Mivel a böngésző a dalszöveg fölé elhelyezett akkordokat is szövegként kezeli, ezért kisebb felbontás esetén először ezt a sort töri meg, majd csak ezt követően a dalszöveget. Végeredményképp a felhasználónak váltakozva jelenik meg két sor akkord majd két sor dalszöveg.

A dolgozat először a projekt követelményeit mutatja be az 1. fejezetben, azután a fontosabb funkcionalitásait ismerteti, majd szemlélteti az architektúráját. Ezt követően a 3. és a 4. fejezet a szerver, illetve kliens oldalon használt technológiákkal foglalkozik. Az 5. fejezet leírja a fejlesztés alatt használt eszközöket és módszereket. Végezetül a 2. fejezet kitér a megvalósítás fontosabb részleteire, valamint a 6. fejezet ismerteti az alkalmazás működését és a továbbfejlesztési lehetőségeket.

A projekt fejlesztése a 2017-es Codespring Mentorprogram részeként, a szakmai gyakorlat alatt indult háromfős fejlesztői csapattal: Bartha Réka, Erőss Ervin és Péter Adorján–Ferenc. A 2017-2018-as tanév első félévében, a csoportos projekt tantárgy keretén belül, egy egyetemi félév erejéig a csapat kibővült két új taggal, név szerint Molnár Örs Hunorral és László Norberttel. A projekt létrejöttéért köszönet illeti a Codespring által kijelölt mentorokat: dr. Simon Károlyt, drd. Sulyok Csabát, Fazakas Tibort, Sebestyén Balázst és Szabó Zoltánt.

1. A Taboo projekt

A Taboo egy webes alkalmazás, mely felhasználóbarát felületet biztosít akkordmenetek szerkesztésére és menedzsmentjére. A következő rész ismerteti a rendszer fontosabb funkcionalitásait, vázolja a használati eseteit illetve a szoftver szerkezetét.

1.1. Követelmények és fontosabb funkcionalitások

Az alkalmazás felhasználói különböző szerepkörökbe sorolhatóak: vendég, bejelentkezett, moderátor és adminisztrátor felhasználó. A szerepkörök által elért funkcionalitások hierarchikusan vannak felépítve. A vendégfelhasználó számára csak néhány funkcionalitás érhető el, míg a bejelentkezett felhasználó rendelkezik a vendégfelhasználó összes funkcionalitásával, valamint további, az előző szerepkör számára nem elérhetőekkel is. Az adminisztrátor felhasználó képezi a hierarchia csúcsát, számára elérhető bármelyik funkcionalitás.

A vendégfelhasználó funkcionalitásai

Amint az 1. ábrán is látható, a vendégfelhasználó számára érhető el a legkevesebb funkcionalitás, ezek közül néhány fontosabb:

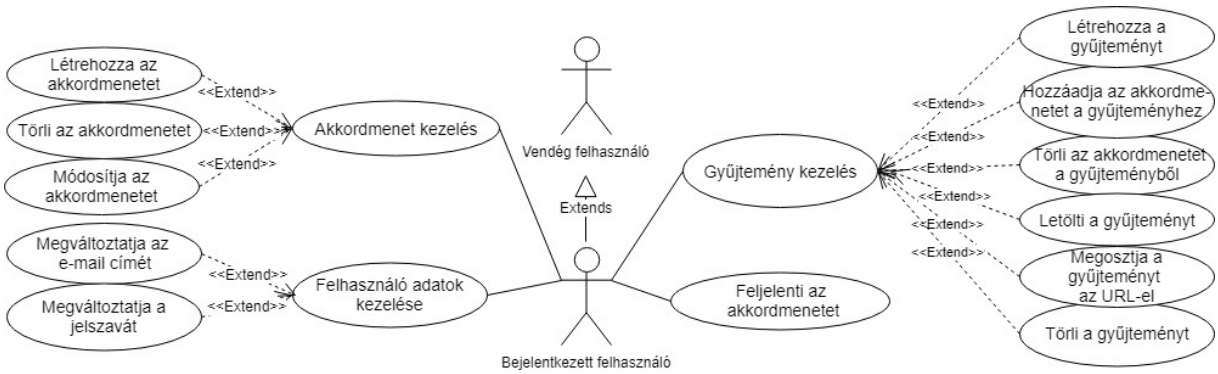
- feltöltött akkordmenetek listázása oldalanként;
- akkordmenetek transzponálása és letöltése PDF formátumban;
- keresés az akkordmenetek között.

A bejelentkezett felhasználó funkcionalitásai

Egy bejelentkezett felhasználónak a vendégfelhasználó által elért funkcionalitások mellet, ahogyan a 2. ábra is szemlélteti, további funkcionalitások válnak elérhetővé. Ezek közül néhány fontosabb:

- akkordmenet létrehozása;
- saját akkordmenet utólagos módosítása és törlése;

1. ábra. A vendégfelhasználó számára elérhető funkcionalitások



2. ábra. A bejelentkezett felhasználó számára elérhető funkcionalitások

- más felhasználó által létrehozott akkordmenet jelentése;
- gyűjtemények létrehozása, törlése és letöltése PDF formátumban;
- akkordmenetek gyűjteményekbe való rendezése;
- gyűjtemények megosztása URL segítségével;
- profiladatok megtekintése és változtatása.

A moderátor felhasználó funkcionalitásai

A moderátor szerepkörrel rendelkező felhasználók számára a következő fontosabb funkcionalitások válnak elérhetővé:

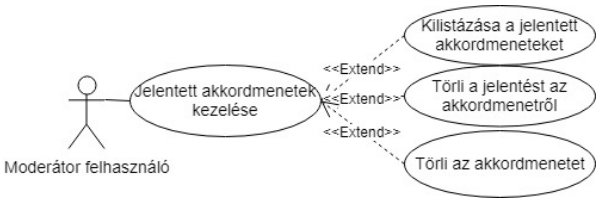
- bármelyik akkordmenet törlése;
- ha nem ő hozta létre az akkordmenetet, akkor erről a jelentés törlése (különben csak egy másik hasonló jogosultságú felhasználó bírálhatja így).

Ezek a funkcionalitások a 3. ábrán tekinthetőek meg.

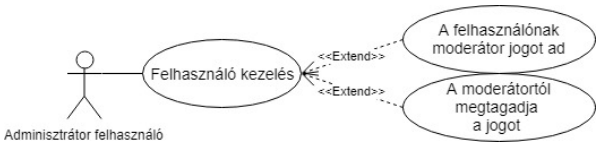
Az adminisztrátor felhasználó funkcionalitásai

Az adminisztrátor elérheti az eddig felsorolt funkcionalitásokat és emellett, ahogyan a 4. ábra is mutatja, igénybe veheti a következőket:

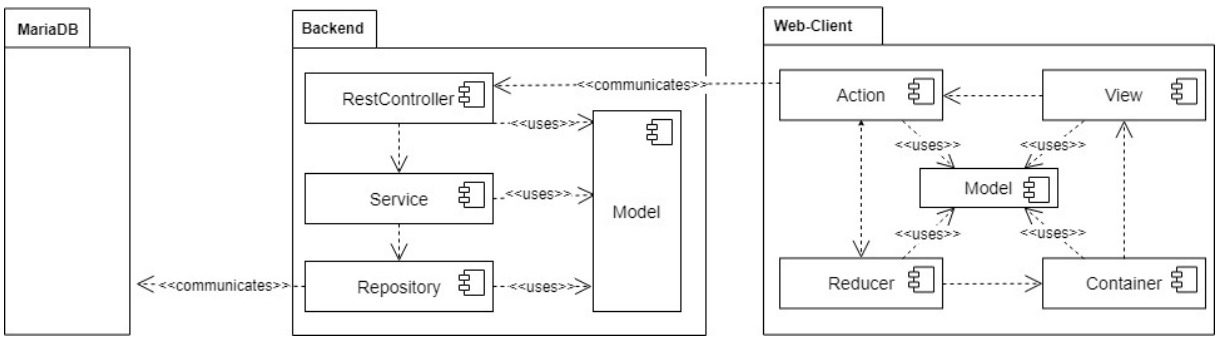
- regisztrált felhasználót moderátor szerepkörrel felruházni;
- moderátortól elveni a moderátori szerepkört (lefokozni egyszerű regisztrált felhasználóvá).



3. ábra. A moderátor számára elérhető funkcionalitások



4. ábra. Az adminisztrátor számára elérhető funkcionalitások



5. ábra. Az alkalmazás architektúrája

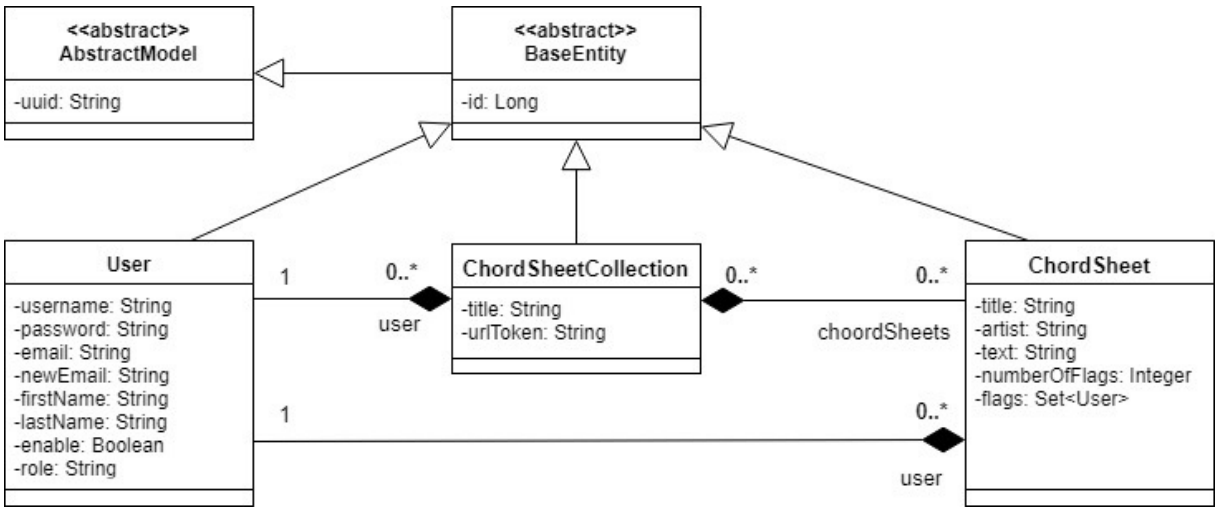
1.2. Architektúra

A Taboo szoftverrendszer három jelentős komponensből tevődik össze (5. ábra): egy kéréseket küldő webes kliensből, ezeket a kéréseket kiszolgáló szerverből és egy adatbázis-kezelő rendszerből.

A kéréseket fogadó és kiszolgáló *Backend* rétegei a *Model*, *Repository*, *Service* és *RestController*, egyénileg jól meghatározott funkciókat töltenek be.

A *Model*-ben vannak eltárolva a szoftverben használt és adatbázisra leképezett adatmodellek, ezek elérhetőek a szerver különböző rétegeiből. A *Repository* réteg valósítja meg a kommunikációt az adatbázissal, valamint felelős a különböző adatbázisműveltekért: beszúrás, módosítás, lekérés és törlés. Az alkalmazás az adatok tárolására és gyors elérésére a MariaDB[21] nevű relációs adatbázis-kezelő rendszert használja. A *Service* rétegen keresztül érhetőek el az alkalmazáson belüli szolgáltatások, itt van megvalósítva az alkalmazás logikája, a *Repository* rétegtől kapja az adatokat és kérésre továbbítja a komponenseknek. A *RestController* egy REST API-t valósít meg, ide érkeznek a kliensektől a REST típusú kérések, valamint ez a komponens felelős a kérések kiszolgálásáért. Megvalósítja az adathozzáférési réteget, azaz kapcsolódik egy relációs adatbázishoz, kérésre létrehozza, törli, módosítja vagy lekéri az adatot. Itt valósul meg a regisztrációhoz illetve elfelejtett jelszó esetén új jelszó megadásához szükséges e-mail küldés, az akkordmenetek illetve gyűjtemények letöltéséhez szükséges PDF generálás. Ezenkívül a felhasználók belépését, kilépését, jogosultságait ellenőrzi, ezáltal korlátozza az adatokhoz való hozzáférést, és biztosítja az adatok védelmét.

A *Web-Client* modul felelős a webes felület felépítéséért és kezeléséért. Az ide tartozó komponensek a következő rétegekből tevődnek össze: *View*, *Action*, *Reducer* és *Container*. A *View* réteg tárolja a megjelenítendő elemek szerkezetét. A *Model*-ben található adatstruktúrákat a rendszer a rétegek közötti adatküldésre használja. Az *Action* réteg szerepe, hogy felhasználói események bekövetkezésére kérést küldjön a szervernek, majd a válasz megérkezése után eseményt generáljon. A *Reducer* réteg felelős ezen események kezeléséért, majd az itt megkapott adatok továbbításáért. A *Container* réteg szerepe, hogy az alkalmazás állapotát frissítse az ide



6. ábra. A főbb szerver oldali entitásokat bemutató osztálydiagram

továbbított adatokkal, ezáltal biztosítva az aktuális tartalom megjelenítését.

1.3. Környezeti elemzés

A szerver entitásait reprezentáló osztályok POJO-k (Plain Old Java Object), amelyek privát adattagokkal, publikus konstruktorral és publikus getter-setter metódusokkal rendelkeznek. Ezek az osztályok az `edu.codespring.taboo.model` csomagban találhatóak, elérhetőek a szerver különböző rétegeiből, valamint kapcsolat írható fel közöttük, amelyet a 6. ábra szemléltet.

Az `AbstractModel` osztály egyetlen adattaggal rendelkezik, amely egy globálisan egyedi azonosítót biztosít az entitásoknak. Ezt kiterjeszti a `BaseEntity`, amely szintén egyetlen adattagot tartalmaz az id-t, amely a relációs adatbázis elsődleges kulcsával egyezik meg. Ezeket az osztályokat terjeszti ki a `User` osztály, ami a felhasználókat, a `ChordSheet` osztály, ami az akkordmeneteket, illetve a `ChordSheetCollection`, mely a gyüjtményeket reprezentálja.

2. A megvalósítások fontosabb részletei

A projekt fejlesztése során számos funkcionalitás valósult meg. A következő része a dokumentációnak bemutatja ezek közül a fontosabbak működési elvét, felhasználva erre ábrákat, kódrészleteket és magyarázatokat.

2.1. Szerver oldali megvalósítások

A Taboo projekt szerver oldali komponense számos fontos funkcionalitásért felelős. Többek között itt valósul meg az akkordmenetek és gyűjtemények letöltéséhez szükséges PDF formátumú állományok generálása, az alkalmazás által kiküldött e-mailek generálása és továbbítása, valamint ez a komponens felelős a kliens oldalról érkező kérések kiszolgálásáért és különböző adatbázis-műveletek végrehajtásáért. A következő rész ezeknek a funkcionalitásoknak a megvalósításait ismerteti.

2.1.1. Adatmodellek

Ahogyan az 1.3. Környezeti elemzés című alfejezetben már említésre került, a szerver központi modelljeit az edu.codespring.taboo.model csomagban található POJO osztályok alkotják. Ezeknek az entitásoknak az adatbázisba való leképezése a JPA(Java Persistence API) specifikáció szerint történik. A JPA specifikáció határozza meg a Java alapú programok objektumainak relációs adatbázisra való leképezését. Ebből kifolyólag az osztályok, amelyeket JPA entitásoknak is nevezünk, többek között JPA annotációkat is tartalmaznak.

A JPA annotációk segítségével megkötések állíthatóak be az adatbázisban szereplő táblákra és azok oszlopaira. Ilyen megkötés lehet például a tábla egyedi kulcsának a megjelölése, amely megadható az *@Id* annotációval, továbbá ennek a generálási típusa is beállítható, ami jelen esetben az *@GeneratedValue(strategy = GenerationType.AUTO)* annotációval történik. A táblák egyes oszlopaira alkalmazva van számos más különböző megkötés. Ilyen például *@NotBlank*, *@NotNull*, vagy a *@Column(unique = true, length = MAX_LENGTH)* annotá-

```
@ManyToMany
    @JoinTable(
            name = "collection_chord_sheet",
            joinColumns = @JoinColumn(name = "chord_sheet", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "collection", referencedColumnName = "id"),
            uniqueConstraints = {
            @UniqueConstraint(columnNames = { "chord_sheet", "collection" })})
    private List<ChordSheet> chordSheets;
```

1. kódrészlet. Gyűjteményeknek és daloknak megfelelő táblák összekötése annotációk segitségével

ció. Az utóbbi használatával egyediség és maximum hosszúság állítható be az adott oszlopra. Ezek mellett JPA annotációk segítségével megadhatóak a táblák között fennálló viszonyok. A táblák között meghatározhatóak *@OneToOne* (1:1-hez kapcsolat), *@OneToMany* (1:n-hez kapcsolat), *@ManyToOne* (n:1-hez kapcsolat) és *@ManyToMany* (m:n-hez kapcsolat) viszonyok. Az 1. kódrészletben látható, hogy a projektben hogyan történik a dalok és a gyűjtemények összekapcsolása annotációk segítségével. A *@ManyToMany* paramétereivel beállításra kerül a kapcsolattábla neve, a kapcsolattartó oszlopok nevei (jelen esetben a táblák elsődleges kulcsai), továbbá egy megkötés, ami egyediséget biztosít a dal-gyűjtemény párosra, mivel egy dal egy gyűjteményben csak egyszer szerepelhet.

Az osztályok felépítésében sor került más keretrendszerek által bevezetett annotációk használatára is, például a *@Data* annotációra, amit a Lombok[23] könyvtár hoz magával. Ez az annotáció megtalálható mindegyik osztály deklarálása fölött, hatására fordítási időben automatikusan kigenerálódnak az osztályok toString(), equals() valamint ezekben az osztályokban lévő attribútumok getter és setter metódusai. Egy hasonló annotáció, amelyik használatra került, a *@JsonIgnore*. Ezt az annotációt a Spring Web MVC által használt Jackson keretrendszer hozza magával, amely a JSON formátumú üzenetek szerializációját és deszerializációját valósítja meg. A Taboo projektben ez az annotáció a password attribútumon található meg. Hatására az alkalmazás API-ja által kiküldött üzenetekbe nem kerül bele a jelszó.

2.1.2. RESTful webszolgáltatások

A Taboo webes applikáció architektúráját tekintve egy SPA (Single-Page Application) alkalmazás. Ez annyit jelent, hogy az oldal megnyitásakor a kliensnek egyetlen egy HTML forrásállomány küldődik ki, a hozzá tartozó állományokkal (JavaScript, CSS, az oldalon szereplő képek, stb.). A tartalmak változásáért ezután a JavaScript program blokkok a felelősek. Ezek a program blokkok az aktuális tartalom megjelenítéséért kéréseket küldenek a szervernek, amelyek válaszként visszaküldik a kért adatokat. Az adatok feldolgozásával és megfelelő sablonba ágyazásával biztosítható, hogy a felhasználónak mindig az aktuális tartalom jelenítődjön meg az oldal újratöltése nélkül. Ez a felépítés feloldja a szerver és kliens közötti szoros függőséget, így akár a teljes kliensalkalmazás átírható a szerver változtatása nélkül vagy fordítva. Emellett csökkenti a szerver és kliens közötti adatforgalmat, mivel az oldal tartalmának változtatásához a megjelenített adatokon kívül nincs szükség újabb állományok kiküldésére.

Az előbb említett előnyök kihasználásának érdekében, a Taboo projektben szükség volt egy API réteg megírására, amely fogadja a kliens oldali kéréseket és kiszolgálja ezeket. Az ide érkező kérések és válaszok HTTP kommunikációs protokollt használnak és szigorúan betartják a REST konvenciókat, így alakítva ki egy RESTful API-t az alkalmazás számára. Ezen réteg implementációja az edu.codespring.taboo.api csomagban található, valamint a Spring Web

keretrendszer(3.1.5) nyújtotta előnyök kihasználásával épül fel. Ezen előnyök egyike, hogy a keretrendszer megoldja a szerverhez érkező kérések szétosztását, így az endpointokra érkező kérések egyszerűen a *@RequestMapping* annotáció használatával összeköthetőek a megfelelő függvényekkel, például az ”/api/chordSheets” endpointra érkező POST kérések hatására végrehajtódik az addNewSheet() függvény. Ez a függvény a kérés paraméteréből, a *@RequestBody* annotációt használva, kiolvas egy ChordSheet-et és ezt beszúrja az adatbázisba. Az ”/api/chordSheets/{id}” URI-ra érkező GET kérés következtére, a szerver visszaküldi az azonosítónak megfelelő akkordmenetet. Ha az előbb említett URI-ra PUT kérés érkezik, akkor hatására az azonosító által meghatározott akkordmenet módosítva lesz a törzsben megadott adatok szerint. Ugyanezen endpointra érkező DELETE kérés esetén a megadott azonosítóval rendelkező akkordmenet törlődik az adatbázisból. Úgy a beérkező, mint a kimenő üzenetek törzsében lévő paraméterek JSON formátumúak, melyeknek a szerializációját és deszerializációját a Spring Web keretrendszer oldja meg. A kiküldött válaszok (a műveletek végrehajtásának függvényében), a megfelelően beállított HTTP állapotkódokat (status codes) tartalmazzák, amely szintén a REST konvenció része.

2.1.3. PDF generálás

Manapság számos szolgáltatás az internet használatán alapul, néhány kattintással bármi megtalálható. Ennek az a veszélye, hogy internet kapcsolat nélkül kizárva maradunk az online világból és ezáltal elérhetetlenné válnak a keresett tartalmak.

Ennek kiküszöbölése érdekében a Taboo lehetőséget biztosít úgy az akkordmenetek, mint a gyűjtemények PDF formátumban való letöltésére, így a kedvenc dalok akkor is megtekinthetőek maradnak, ha nincs internetkapcsolat. Mindemellett a generált PDF könnyen olvasható és nyomtatóbarát, nem utolsó sorban az is megemlítendő, hogy a gyűjtemények nyomtatásával bárki létrehozhat saját kezűleg válogatott zenefüzeteket.

A Taboo projektben a PDF generálása szerver oldalon történik. Ez a művelet a Flying Saucer(3.3) függvénykönyvtár által megírt ITextRenderer osztály segítségével valósul meg. A generálandó állomány tartalma megadható az osztály setDocumentFromString(String content) metódusával, mely paraméterként egy XHTML-t tartalmazó karakterláncot vár. Az említett paraméter előállításához feldolgozásra kerül a DSL (Domain Specific Language) által leírt akkordmenet. A projektben az akkordmenetek leírására meghatározott DSL szerkezete, ahogyan a 7. ábrán is látható, abból áll, hogy az akkordok kapcsos zárójelek között bekerülnek a dalszövegbe a nekik megfelelő helyen. Ezeknek a dalszövegeknek a feldolgozása annyit jelent, hogy a kapcsos zárójelekben lévő akkordok CSS osztályok hatására a szöveg fölé helyeződnek el, így generálva a kívánt külalakot. A tartalom megszerkesztése után, az ITextRender osztály createPDF(OutputStream os) függvényével a PDF állomány kiírható a paraméterként megadott

kimenetre. Ez a metódus a háttérben az iText függvénykönyvtárat használja. Mivel a szerveren nincs igény a generált állományok tárolására, ezért ezeknek a kiírása temporális fájlokba történik, amelyek csak a használat idejéig foglalják a memóriát.

2.1.4. E-mail küldés

A projekt célkitűzései között szerepel, hogy felhasználóinak biztosítsa azokat az elvárt alapfunkcionalitásokat, amelyek hiányai felfedezhetőek számos ehhez hasonló webes alkalmazásban. Ezen hiányosságok javításának érdekében a Taboo oldal támogatja az elfelejtett jelszó megváltoztatását.

Annak érdekében, hogy ezt a funkcionalitást felkínálja az oldal, szükséges bevezetni a kétlépéses regisztrációt. Ez abból áll, hogy regisztráció esetén a személyes adatok megadása után a szerver egy egyedi azonosítót generál, amelyet egy link paramétereként a felhasználó által megadott e-mail címre küld. A linkre kattintva, a felhasználó bizonyítja, hogy az e-mail cím valóban a tulajdonában áll, így belépést nyer a rendszerbe.

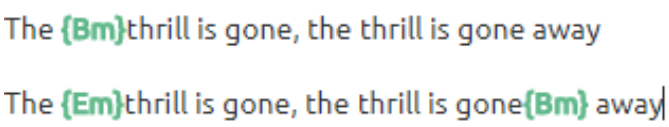
Elfelejtett jelszó esetén a felhasználó az e-mail cím megadásával jelezheti, hogy új jelszót szeretne hozzárendelni a fiókjához. Ennek következtében egy linket küld a rendszer a felhasználó e-mail címére, amely egy űrlaphoz navigálja, ahol megadhatja új jelszavát. A szerver az egyedi azonosítót használva frissíti a felhasználó jelszavát.

A regisztráció és elfelejtett jelszó problémáját megoldó folyamatokhoz hasonlóan, a bejelentkezett felhasználónak módjában áll a regisztráláskor megadott e-mail címét megváltoztatni. Egy ellenőrző e-mail segítségével biztosítjuk az újonnan megadott e-mail cím érvényességét.

A fentebb ismertetett funkcionalitások megvalósításához elengedhetetlen az e-mail küldés. Ennek kivitelezésére a projekt a Mailgun szerver szolgáltatásait használja, amihez szükséges a JavaMailSenderImpl osztály egy példányát konfigurálni, hogy az csatlakozzon a Mailgun szerverhez. A Taboo projektben ez a beállítás az edu.codespring.taboo.config csomagban található MailConfiguration nevű konfigurációs osztályban valósul meg. Ez az osztály a mail.properties fájlból kiolvassa a szükséges paramétereket és beállítja azokat a JavaMailSenderImpl osztály példányának. A levelek tartalmának generálása az FTL (FreeMarker Template Language)(3.2) sablonok adatokkal (felhasználónév, link, aktuális dátum) való kitöltésével történik. A generálás után a JavaMailSenderImpl osztályban található send() metódus továbbítja az e-mailt a Mailgun szervernek, amely kézbesíti a felhasználóknak.

2.2. Kliens oldali megvalósítások

Ez a fejezet, a kliens oldal fontosabb funkcionalitásainak megvalósításának részleteivel foglalkozik, mint például a szerverrel való kommunikáció vagy az akkordmenet szerkesztő.



7. ábra. Az akkordmenet tárolására és kezelésére használt DSL formátuma

2.2.1. Akkordmenet szerkesztő

A Tabbo projekt fő funkcionalitásai közé tartozik akkordmenet szerkesztő, mely célja lehetővé tenni a weboldal tartalmának a bővítését és karbantartását. A felület fejlesztése közben különös figyelmet kapott, hogy a szerkesztési folyamat egyszerű és magától érthető legyen a felhasználónak. A szerkesztő kétféle akkordbevitelt támogat: interaktív és nyers formátumot. A legnagyobb kihívást az interaktív akkordbevitel jelentette, ahol a felhasználó egér segítségével navigálhat a dalszöveg sorai között és a kívánt pozícióba kattintva szúrhat be akkordokat vagy törölhet másokat.

A szerkesztő meg kell tudja különböztetni a dalszöveget a bevitt akkordtól, ennek érdekében lett megalkotva egy DSL (Domain Specific Language). Ahogy a 7. ábrán is látható a kapcsoszárójel közé tett betűket a szerkesztő értelmezője úgy ismeri fel, mint egy akkord. Az akkordmenet adatbázisba való mentése, az oldalon való megjelenítése és szerkesztése is a DSL felhasználásával történik. Az oldalon való megjelenítés, az alkalmazás által definiált feldolgozó egység által történik. Az alapötlet az volt, hogy az adatbázistól kapott szöveget feldarabolja soronként, majd akkordonként és az így kapott részeket HTML címkékbe ágyazza, ezáltal CSS stíluselemek segítségével formázható a megjelenítendő akkordmenet. Minden akkordnak a DSL jóvoltából meg van a pontos helye a lementett dalszövegben és mindig megjelenítéskor alkalmazzuk rá a formázást, ebből következik, hogy kisebb kijelzőn, mint például telefonon, sortörés után is fogja tartani a pozícióját.

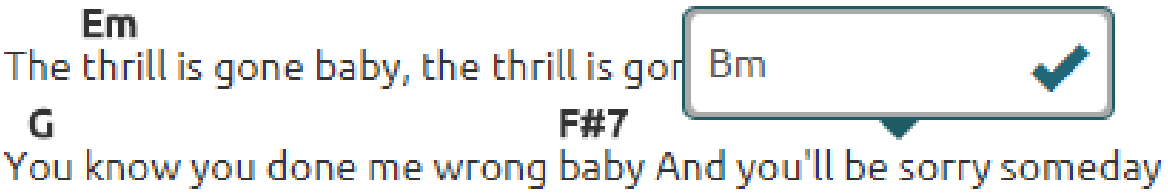
Az akkormenet szerkesztője a Facebook által fejlesztett, Draft.Js(4.4) szövegszerkesztő keretrendszer felhasználásával lett megalkotva. Draft.Js a *contenteditable* HTML attribútum React(4.1) filozófia szerinti felhasználása. A keretrendszerrel létrehozott szerkesztőnek hierarchikus felépítése van. A legfelső szinten található az *EditorState* JSON objektum mely tartalmazza a szerkesztő teljes állapotát. Ez alatt helyezkedik el az úgynevezett *ContentState* amely képviseli a szerkesztőben található tartalom teljes állapotát, a beírt szöveget továbbá az erre alkalmazott stíluselemek és entitások hatásköreit. A *ContentState* egy rendezett adatstruktúrában tárolja a *ContentBlock* objektumokat, ezek képviselik a szerkesztő sorainak a teljes állapotát. Az *Entity* egy statikus modul, ami tartalmaz egy API-t entitás objektumok készítésére, módosítására és törlésére. Ezek az objektumok teszik lehetővé szöveg részek metaadatokkal való felruházását. Amint a 7-es ábrán is látható az akkordokra más formázás van alkalmazva, ez azért válhat valóra

```
export const RawChordSpan: React.SFC<Props> = (props: Props) => {
    return (
        <span
            className="raw-chord"
            data-offset-key={props.offsetKey}>
            {props.children}
        </span>
    )
}
```

2. kódrészlet. React komponens a DSL "díszítéséhez"

mert a szerkesztő értelmezi a szöveghez kapcsolt metaadatot és egy speciális React komponenst illeszt díszítő elemek (*Decorator*) segítségével az idetartozó részre. A "dekorátor" rendszer fogalma azon alapszik, hogy a keretrendszer vizsgálja egy adott *ContentBlock* tartalmát és ha van szövegrész ami illeszkedik egy meghatározott stratégiához, a 2. kódrészletben látható React komponenshez hasonlóval "feldíszíti" azt. A stratégiák lehetnek egyszerű reguláris kifejezések vagy a szövegrészekhez rendelt entitások is.

Kiemelt figyelmet kapott az interaktív akkordbevitel, mivel az alkalmazás nem akarja rákényszeríteni a felhasználót a DSL nyelv szintaxisának a megtanulására valamint a jelölőnyelv változtatásainak a követésére. Ez úgy valósúl meg, hogy egérrel való kattintás hatására egy ablak jelenik meg a pozíció fölött. Az ablak szerkezete egyszerű, amint a 8. ábrán is látszik, tartalmaz egy beviteli mezőt és egy gombot, amivel véglegesítheti a műveletet. Kihívást jelentett megtalálni a megfelelő módszert az ablak pozicionálására és mozgatására, mivel hogy változó kijelzőméreteknek megfelelően kell meghatározni a pontos helyzetét az egérmutató fölött. A szerkesztő interaktív módon támogatja az akkordok beszúrását, módosítását valamit törlését. Ezek a funkcionalitások a Draft.Js keretrendszer moduljai segítségével vannak implementálva. Minden módosítás beillesztése előtt az akkordon egy ellenőrzést végez el a szerkesztő. A kezdő karaktereket ellenőrzi, hogy előfordulhatnak-e, mint akkord kezdődés, például „W” betűvel nem kezdődhet egy akkord, míg „A” betűvel igen. Ha be van kapcsolva az interaktív mód akkor a bevitt akkordok könnyedén, egy kattintással szerkeszthetőek, mivel mindig ellenőrzi azt, hogy adott pozícióban található-e entitás, aminek szövegét betölthetné az előugró ablak beviteli mezőjébe. A nehézséget az jelentette, hogy az entitásnak megfelelő szövegrészt kimásolja a



8. ábra. Az akkordok interaktív beszúrását lehetővé tevő ablak

```
export interface Action<T> {
    type: string;
    payload?: T;
}
```

3. kódrészlet. A Redux események szerkezete

teljes tartalomból mivel a metaadatokról a szerkesztő csak a soronkénti karakter eltolás számát tárolja.

Egy akkordmenet későbbi szerkesztésénél az adatbázistól kapott DSL formátumú szövegből fel kell építeni a szerkesztő *ContentState*-jét. Ezt az alkalmazásban definiált segédfüggvény a **function convertFromDSL(dsl: string)** végzi, ami feldarabolja a szöveget és belőle létrehozza a megfelelő objektumokat, amiket ezután a szerkesztő állapotához ad.

2.2.2. Kommunikáció a szerverrel

A kliens és szerver kapcsolata REST alapú kérésekkel és válaszokkal valósul meg JSON formátumban. Az alkalmazás kliens oldala a React Javascript keretrendszer felhasználásával készült. Ez ki van egészítve Redux(4.2) állapotmenedzserrel annak érdekében, hogy az alkalmazás működését átláthatóvá tegye és későbbi karbantartását, valamint továbbfejlesztését megkönnyítse. Ennek segítségével megvalósítható, hogy az adatok egy irányban áramoljanak és előre meghatározható, hogy az alkalmazás állapota hogyan fog változni.

Az alkalmazásnak egy úgynevezett tárolója van, ami tartalmazza a teljes állapotát. Ebben az adatok csak olvashatóak és minden változtatást események kiváltásával lehet eszközölni. Ezeket az eseményeket *Reducer*-k hajtsák végre, amik tiszta mellékhatás nélküli függvények. A tiszta függvények nem függnek külső állapottól mint például adatbázistól, ugyanarra a bemenetre mindig ugyanazt az eredmény produkálják. A szerver fele irányuló kérések a Redux Thunk köztes réteg segítségével vannak lebonyolítva, ez lehetővé teszi az aszinkron végrehajtást.

Példaként megfigyelhető az akkordmeneteket megjelenítő komponens működése. A felhasználó rákattint a megjelenítendő akkordra a listában, ezután a *Router* objektum megkeresi a neki megfelelő komponenst és felépíti azt. Minden React komponensnek van egy életciklusa. Az életciklus minden szakaszának megfelel egy-egy metódus, amelyekkel testre szabható. Amint felépült a komponens azután jöhet a tartalom, amit a felhasználó szeretne látni, ennek érdekében

```
public componentDidMount(): void {
        this.props.fetchSheet(
            this.props.match.params.id, this.props.role, this.props.userPrefix);
}
```

4. kódrészlet. A React komponens betöltődés utáni állapotához tartozó függvény

```
export const sheetDetails = (state: Sheet = initialState, action:Action<Sheet>):
Sheet | undefined => {
    switch (action.type) {
        case 'FETCH_SHEET_SUCCESS':
            return action.payload;
        default:
            return state;
    }
};
```

5. kódrészlet. Az akkordmenetnek megfelelő Reducer függvény

a 4. kódrészletben látható speciális függvényben kiváltódik egy esemény. Ezek típusa, amint látható a 3. kódrészletben is, egyszerű általános Typescript interfészek, ezáltal elkerülhetőek az ismétlődő objektum definíciók. Az esemény, a komponensnek úgynevezett tulajdonságként továbbadott, általánosan *ActionCreator*-nak nevezett függvények segítségével hozható létre.

Mivel ebben az esetben egy kérést kell küldeni a szerver felé, így egy olyan esemény váltódik ki amely egy függvényt térít vissza. Ez a függvény építi fel és küldi el a szervernek a kérést. Ezután a szervertől kapott válasz függvényében létrehozza a megfelelő eseményt az alkalmazás tárolójában található adat módosítása érdekében. Így megoldható az aszinkron végrehajtás. Ezt követően az alkalmzásban definiált *Reducer*-k lépnek működésbe, amik a kiváltott *Action* típusa alapján társítanak egy a tárolóban található objektumot az újonnan érkező adatnak. A rendszer minden *Reducer*-t leellenőriz, hogy található-e megfelelő típus, ezért annak érdekében, hogy az adat ne vesszen el, az aktuális függvény mindig vissza kell térítse a kapott értékeket. Az 5. kódrészletben látható a felhasználó által kért akkordmenet objektumnak megfelelő *Reducer*. Amint a változtatás megtörtént, a rendszer minden az adott objektumtól függő komponenst frissít azáltal, hogy továbbadja nekik az új értékeket két speciális függvény segítségével. A 6. kódrészletben látható egy **mapStateToProps** és egy **mapDispatchToProps** nevű függvény. Az első függvény lemásolja a tárolóból a komponens számára, azokat az objektumokat amelyekre szüksége van és ha változás történt, akkor frissül az adott kinézet ezek függvényében. A második függvény az *ActionCreator* metódusokat felelteti meg a komponensnek. Miután sikeresen megkapta a komponens az akkordmenetet a fenti függvények segítségével, frissül és megjeleníti ezt.

```
const mapStateToProps = (state: State, ownProps: ContainerProps): State => ({
    sheetDetails: state.sheetDetails,
});

const mapDispatchToProps = (dispatch: Dispatch<State>): {} => ({
    fetchSheet: (id: number, role: string, userPrefix: string): void = {
            fetchSheet(id, role, userPrefix, dispatch) },
});
```

6. kódrészlet. A tárolóban történő változásokat a komponensnek megfeleltető függvények

3. Szerver oldali technológiák

A Taboo projekt a Spring keretrendszer komponenseinek használatával definiálja a különböző architektúrális rétegeit. A keretrendszer számos funkcionalitást tartalmaz, amelyeket a fő tulajdonságaik alapján a következő csoportokra lehet osztani: központi konténer, adat hozzáférés/integráció, Web, AOP (Aspect Oriented Programming), üzenetküldés, teszt és egyéb eszközök. Ebben a fejezetben a projekt kivitelezéséhez használt technológiák kerülnek részletes bemutatásra.

3.1. Spring

A Spring [26] pehelysúlyú keretrendszer egy átfogó konfigurációs és programozási modell, a Java alapú vállalati alkalmazások fejlesztéséhez. A pehelysúlyú jelleg a keretrendszer moduláris felépítésében mutatkozik meg, a különálló komponensek csak akkor kerülnek beillesztésre, ha ténylegesen szükség van rájuk. A Spring keretrendszer magját az IoC (Inversion of Control[11]) konténer képezi, amelynek egy specifikus esete a dependency injection (DI) tervezési minta. Komplex Java alkalmazásban definiált osztályok általánosak és egymástól amennyire csak lehet függetlenek kell legyenek, növelve az újrafelhasználhatóság esélyét. Ezen osztályok függőségeit és életciklusát felügyeli az IoC konténer. Ezt a Java nyelv Reflection mechanizmusa által tudja megvalósítani.

Az adatbázis hozzáférési réteg a Spring JPA (3.1.2) segítségével van megvalósítva. Spring Security (3.1.3) az alkalmazás biztonságáért felelős, míg a Spring Social Facebook (3.1.4) az alkalmazás és a Facebook kapcsolatát teszi lehetővé. A Spring Web (3.1.5) az alkalmazás RESTful API-jának megvalósításához járul hozzá. A különböző felhasználónak küldött email-ek, mint például a regisztráció megerősítése, az Apache FreeMarker és JavaMail (3.2) segítségével vannak létrehozva. A PDF dokumentumok generálása a Flying Saucer és iText könyvtárak (3.3) felhasználásával vannak implementálva.

3.1.1. Spring Boot

A Spring Boot [22] célja megkönnyíteni a Spring alapú alkalmazások létrehozását. Ennek érdekében eszközöket biztosít és előkonfigurált Spring modulokat tartalmaz. A Spring Boot lehetővé teszi rendszer paraméterek külső fájlban való megadását. Ezen fájlok kiterjesztése lehet például *properties* vagy *YAML*. A Taboo alkalmazás a *YAML* formátumot használja. Beépített Apache Tomcat [1] szervlet konténert is biztosít, ebből kifolyólag az alkalmazás futtatása érdekében nem szükséges egy külső alkalmazásszerver beállítása.

3.1.2. Spring Data JPA

Adatok tárolására, a Taboo projekt a JPA specifikáció implementációját, a Hibernate[17] Object/Relational Mapping keretrendszert használja. Az Object/Relational Mapping keretrendszer használatával a fejlesztő relációs adatbázisban lévő adatokat objektum orientált paradigmáknak megfelelően manipulálhat. Ilyen paradigmák például a polimorfizmus vagy öröklődés. A Spring Data JPA [13] egy absztrakciót képez az alkalmazás adathozzáférési rétege fölött, ezáltal például a teljes adatbázis hozzáférési réteg lecserélhető, anélkül hogy bármely módosítást kellene végezni a felsőbb architekturális rétegeken. A JPA annotációk alkalmazásával az entitások szintjén automatikusan generálhatóak a megfelelő adatbázis interfészek, amelyek implementációjáról a keretrendszer gondoskodik. A rendszer interfészekben a metódusnév elnevezési konvenciók segítségével generálja a specifikus lekérdezéseket.

3.1.3. Spring Security

Az alkalmazás biztonságáról a Spring Security [3] keretrendszer gondoskodik, ami egy átfogó biztonsági szolgáltatást nyújt a Java alapú vállalati rendszereknek. A keretrendszer a beérkező kéréseket hitelesíti és ellenőrzi. Itt lehet szabályokat megadni, hogy bizonyos szerepkörrel rendelkező felhasználó milyen erőforrásokhoz fér hozzá. A keretrendszer sok fajta hitelesítő modellt támogat, mint például LDAP, űrlap alapú azonosítás vagy HTTP BASIC azonosító fejlécek. A Taboo alkalmazás ezek közül az űrlap alapú azonosítást használja, tehát a felhasználónak a bejelentkezéshez meg kell adnia a felhasználónevét és a jelszavát.

3.1.4. Spring Social Facebook

A Spring Social [7] lehetővé teszi a Java alapú alkalmazások kapcsolatteremtését Software-as-a-Service(SaaS) szolgáltatókkal, mint például a Facebook vagy Twitter. Az ilyen szociális integrációk esetén három fél között kell kapcsolatot teremteni. Ebben az esetben az egyik fél a Facebook mint SaaS szolgáltató, második a Taboo alkalmazás és a harmadik fél a felhasználó aki egy-egy felhasználói fiókot üzemeltet az előző feleknél. A Spring Social Facebook az alap keretrendszer egy kiterjesztése. Az alkalmazás ezt felhasználva csatlakozik a Facebook-hoz. A keretrendszert felhasználva az alkalmazás átruházza a bejelentkező felhasználó azonosítását a Facebook-ra.

3.1.5. Spring Web

A Spring Web keretrendszer feladata a szerver által kapott REST kérések értelmezése és JSON objektumok szerializációja és deszerializációja. A Taboo projekt által definiált RESTful API ezt a modult felhasználva szolgálja ki a beérkező kéréseket. Annotációk segítségével kezelő

függvényeket feleltet meg a REST végpontoknak. Emellett automatikusan alkalmazáson belüli kivétel típusokat térképez HTTP hibakódokra.

3.2. Apache FreeMarker, JavaMail

A Spring keretrendszer szolgáltat egy absztrakciót az e-mail küldés megvalósítására, amely elrejti a felhasználó elől a háttérben lévő e-mail küldő rendszer specifikációját és kezeli az alacsony szintű erőforrásokat. Az e-mailek küldését a JavaMail [19] referencia implementáció segítségével valósítja meg, ami egy platform- és protokoll-független keretrendszer Java alapú alkalmazásokhoz. Az alkalmazás által a felhasználónak küldeni kívánt e-mail az Apache FreeMarker [12] sablon motor segítségével van megszerkesztve. Ez egy Java könyvtár, amely szöveges kimenetet generál egy szakterület specifikus nyelv, a FreeMarker Template Language (FTL) alapján.

3.3. Flying Saucer, iText

Az alkalmazás által kezelt akkordmenetek PDF formátumba való alakítása a Flying Saucer [27] és az iText [2] könyvtárak segítségével történik. A Flying Saucer könyvtár HTML és CSS bemenet alapján generál egy kirajzolt reprezentációt kimenetként. A CSS szabályok lehetnek a HTML dokumentumba beágyazva vagy külső forrásként hozzákapcsolva. Az előzőleg generált Flying Saucer kimenetből az iText könyvtár segítségével, PDF dokumentum generálható.

4. Kliens oldali technológiák

A Taboo kliensalkalmazása egy webes alkalmazás, ahol a megjelenített komponensek Typescriptben íródtak, valamint felhasználnak JavaScript alapú keretrendszereket, többek között a Reactet, Redux-ot és Draft.js-t. Emellett fontos szerepet tölt be a reszponzív design kialakításában a Bootstrap függvénykönyvtár. A dokumentáció a következőkben ismerteti a megvalósításban használt fontosabb technológiákat.

4.1. React

A React [24] egy nyílt forráskódú JavaScript könyvtár, melyet a Facebook fejlesztett ki. Leegyszerűsíti az olyan dinamikus webes alkalmazások készítését, amelyek adatokat dolgoznak fel, jelenítenek meg, valamint események bekövetkezésére frissítik a tartalmat az oldal újratöltése nélkül. Célja, hogy biztosítsa a nagyméretű alkalmazások átláthatóságát, gyorsaságát és egyszerűségét.

Fontos megemlíteni, hogy komponens alapú, ami azt jelenti, hogy a programozó elkülönített, önállóan működő egységeket határoz meg, amelyek tulajdonságokkal illetve állapottal rendelkeznek. Ezek a tulajdonságok a komponensen belül változtathatatlanok, az őt létrehozó úgynevezett szülőkomponens állítja be, valamint élettartamuk megegyezik a komponens élettartamával. Az állapotok tárolják a megjelenítendő adatokat, amelyeknek a változtatásával érhető el a DOM (Document Object Model) elem frissülése. Ezzel a fejlesztő számára egy kényelmes felületet biztosít a DOM elem manipulálására. A React komponensmodellje teljesen eltér a Model-View-Controller (MVC) tervezési mintától, de további keretrendszerek használatával, mint például a Redux, hasonló mintát lehet kialakítani. Az így kialakított MVC mintában a React komponensek felelnek meg a View-nak, a Redux által behozott Reduce-ek a Controller-nek, valamint a TypeScript-ben írott adatmodellek a Model-nek.

A React komponensek jellemzően JSX-be íródnak. Ez egy React specifikus, szintaktikai kiterjesztése a JavaScript-nek, amely leginkább az XML/HTML-hez hasonlít. Ennek a kiterjesztésnek köszönhetően írható HTML kód a JavaScript állományokba. Az előfeldolgozó egységek ezeket a HTML kódrészeket a JavaScript motor számára értelmezhető objektumokká alakítják. Mindez lehetővé teszi a React komponensek egymásba ágyazását, így építve egyre nagyobb és összetettebb egységet. [20]

4.2. Redux

A Redux [25] egy nyílt forráskódú JavaScript könyvtár, amelyet Dan Abramov fejlesztett ki 2015-ben. Általában, de nem kötelezően, együtt használják a React vagy Angular függvény-

könyvtárakkal a felhasználói felület kiépítésére. Segítségével egyszerűbben kezelhető az alkalmazás állapota, mivel kezeli a megjelenített adatokat és a különböző felhasználói műveleteket.

Meghatározhatóak különböző típussal és egy adattaggal rendelkező események, melyek felhasználói műveletek bekövetkezésére kiváltódnak. Ezeket a Reducer-ek, típustól függően elkapják, majd a megadott adattaggal frissítik az alkalmazás állapotát, így aktualizálják a felhasználói felületet. A Reducer-ek határozzák meg, hogy hogyan változzon az alkalmazás állapota a kiváltott események alapján.

4.3. TypeScript

A TypeScript [29] egy nyílt forráskódú, Microsoft által fejleszt objektumorientált nyelv, amely a JavaScript-et bővíti ki, célja megkönnyíteni az összetett, nagyméretű web alkalmazások fejlesztését. A TypeScript kódot egy fordító, a böngészők számára is értelmezhető, JavaScript kódra fordítja.

A nyelv támogatja az enum, generikus típus, osztályok és interface-ek használatát, valamint az öröklődést is. Bevezeti a típus fogalmát, ezáltal elősegíti az adatmodellek pontosabb leírását, így ezek átláthatóbbá válnak, továbbá ezzel segít kiküszöbölni a típus inkompatibilitás okozta problémákat és csökkenti a hibalehetőségek számát.

4.4. Draft.js

A Draft.js [9] egy nyílt forráskódú JavaScript alapú keretrendszer, amelyet a Facebook fejlesztett ki a React függvénykönyvtárhoz. Lehetőséget biztosít olyan szövegszerkesztők megvalósítására, amelyek képesek feldolgozni úgynevezett rich (gazdag, színezett, díszített) szövegeket. A keretrendszer eszközöket szolgáltat, amelyek kiterjeszthetőek és testre szabhatóak. Ezen eszközök segítségével használója felépítheti saját, tetszés szerinti szövegszerkesztőjét. A Draft.Js az immutable-js [18] felhasználásával lett megalkotva, ami egy API-t szolgáltat funkcionális állapot frissítésre. A Taboo alkalmazás akkordmenet szerkesztője ennek a keretrendszernek a felhasználásával lett megalkotva.

4.5. Bootstrap

A Bootstrap [4] függvénykönyvtárat web kliens alkalmazások készítéséhez használják, Mark Otto és Jacob Thornton fejlesztette a Twitter-nél, valamint 2011-től nyílt forráskódú, azaz bárki számára ingyenesen elérhető.

A függvénykönyvtár előre megírt CSS, HTLM illetve JavaScript kódokat tartalmaz, amelyek lehetőséget biztosítanak navigációs sávok, felugró ablakok, legördülő menük és más ehhez

hasonló elemek használatára. Emellett, segítségével megvalósítható a weboldal különböző méretű képernyőkhöz való igazítása (a képernyőméret függvényében elemek méretezhetőek át, tüntethetőek el, mozdíthatóak el), így a kisebb készülékeken is átlátható, használható marad az alkalmazás.

5. Eszközök és módszerek

A szoftverfejlesztés gördülékenysége és a termék minőségi megvalósítása szempontjából elengedhetetlen bizonyos módszerek és segéd eszközök használata. Ilyen módszer például a Scrum vagy a Test-Driven Development. A kódminőség ellenőrző vagy a Continous Integration-t lehetővé tevő eszközök végig kísérik a fejlesztési folyamat fázisait ezáltal folyamatos visszajelzést szolgáltatva a projekt állapotáról. A következő rész az alkalmazás fejlesztése során használt módszereket és eszközöket mutatja be.

5.1. Projekt menedzsment

Az alkalmazás fejlesztése a Scrum szabályai szerint folyt. A Scrum egy agilis módszer, iteratív és inkrementális melynek alapján, a fejlesztés a szakgyakorlat alatt két hetes iterációkban valósult meg, mindennapi megbeszélésekkel. Később a csoportos projekt tantárgy keretein belül 3 hétre nőtt a sprint hossza és hetente egyre csökkent a megbeszélések száma. A Scrum keretein belül az iterációkat sprinteknek nevezik. A sprinteket egy tervezési fázis előzte meg, amelyen a fejlesztők megbecsülték az egyes, úgynevezett story pontok értékét. Story pontokban mérik egy adott funkcionalitás komplexitását. Minden sprint az újonnan fejlesztett funkcionalitások bemutatásával zárult a terméktulajdonos szerepét is betöltő mentorok előtt.

5.2. Verziókövetés, folyamatos integráció és kitelepítés

Az alkalmazás fejlesztése alatt, a Git verziókövető rendszer volt használva, ezáltal nyomon követhetőek maradtak a kódbázisban történő változások és elősegítette a csapatmunkát is. A fejlesztők a projekt funkcionalitásait külön fejlesztési ágakon fejlesztették. Az adott funkcionalitás implementálását szigorú kódvizsgálat követte. A projekt menedzsmentben elengedhetetlen volt a GitLab repository manager használata, ami lehetővé tette feladatok számontartását, a folyamatos integrációt, valamint az alkalmazás kitelepítését. Fejlesztés során a hibák mielőbbi kiszűrése érdekében folyamatos integráció volt használva ami minden *push* művelet után, egy adott fejlesztői ágra, lefordította a projektet és lefuttatta a rendelkezésre álló teszteket. A folyamat utolsó művelete, a fő fejlesztési ágon, a szoftver teszt szerverre való kitelepítése volt. Fejlesztés során az alkalmazás a H2[16] memória adatbázist használta majd kitelepítés után a MariaDB[21] relációs adatbázist. Ezeket a beállításokat a szerver konfigurációs állományában definiált fejlesztői és produkciós profilok tették lehetővé. A fejlesztői és a kitelepítés utáni platformok közötti különbségekből adódó konfigurációs problémák elkerülése érdekében, az alkalmazás Docker segítségével van kitelepítve a szerverre. A Docker[8] rendszer szintű virtualizációt biztosít célja megkönnyíteni alkalmazások kitelepítését és futtatását a konténerek által.

A konténerek lehetővé teszik a fejlesztőknek összecsomagolni az alkalmazást minden függőségével együtt és egy csomagban szállítani, így biztosítva, hogy az alkalmazás futni fog bármely Linux rendszeren.

5.3. Kódminőség ellenőrzés

Fejlesztés során nagyon fontosak a konvenciók és ezek betartása, ezáltal is megkönnyítve a csapatmunkát. Az alkalmazás teljes fejlesztési folyamata során, kódminőség ellenőrző eszközök vigyáztak a konvenciók betartatására, ilyenek a Checkstyle, a Findbugs és a TsLint.

A Checkstyle[6] egy statikus kódminőség ellenőrző eszköz Java nyelvhez. Automatizálja a kód konvenció ellenőrzési folyamatot, megkímélve ettől a fejlesztőket. Egy magas fokú testre szabhatóságot biztosít. Megtalál osztály, vagy metódus design problémák mellett kód formázási és elrendezési hibákat is.

A FindBugs[10] is egy Java kódminőség ellenőrző, de ellentétben a Checkstyle-al nem foglalkozik a formázási vagy kódolási standardokkal, inkább a logikai problémákra és a jobb módszerekre fókuszál, mint például redundáns "null pointer" ellenőrzés vagy figyelmen kívül hagyott visszatérítési érték. Valójában potenciális hibákat és teljesítmény problémákat keres bytecode szinten. Nagyon pontosan jelez nehezen megtalálható hibákat, mint például szál szinkronizálási problémák.

A TsLint[28] egy kiterjeszthető statikus kódminőség ellenőrző eszköz, ami vigyáz TypeScript kódok konvencióinak betartására. Nem csak szintaktikai elemzést végez hanem szemantikait is, mint például megvizsgálja a statikus típusokat és kódolási mintákat.

5.4. Fordítás és függőségek menedzselése

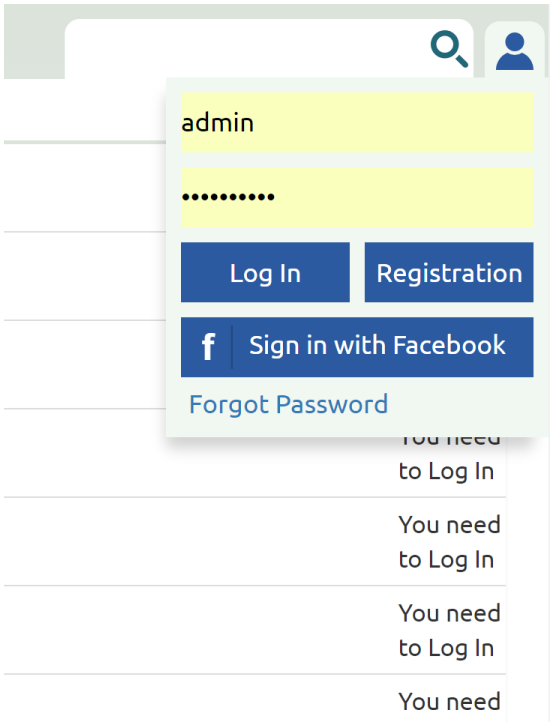
Komplex alkalmazások esetén nélkülözhetetlen az olyan eszközök használata amelyek segítségével automatizálható a fordítási folyamat, valamint a projekt függőségeinek a menedzselése. A Taboo alkalmazás két modulból tevődik össze. Ezeket a Gradle[14] függőség menedzsment és fordítást automatizáló eszköz fogja össze. A *backend*, Java nyelvben íródott modul automatikus fordítását és függőségeinek kezelését a Gradle végzi. A *web* TypeScript nyelvben íródott modul, amely függőségeinek a letöltéséhez szükség volt a Yarn[30] csomagkezelőre. Ennek telepítése egy fordítás közben meghívott Gradle feladat segítségével van biztosítva. A TypeScript-et nem tudják értelmezni a web böngészők, ebből kifolyólag le kell fordítani egy böngészőknek is értelmezhető JavaScript standardra. Ez a fordítási folyamat a Gulp[15] eszköztár segítségével van automatizálva. A folyamat magába foglalja a TypeScript kód JavaScript standardra való fordítását és tömörítését, amelyet a Gulp Browserify[5] kiegészítője valósít meg.

6. A Taboo működése

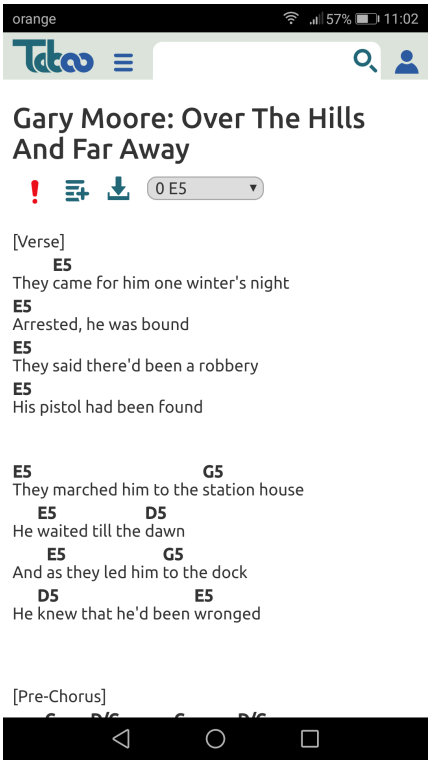
Ebben a fejezetben a Taboo webes alkalmazás lényeges funkcionalitásai kerülnek bemutatásra képek és leírások használata által. A kinézet a tervezésében fontos szerepet játszott, hogy az alkalmazás használható maradjon a kisebb készülékeken is, így a felület felépítésében szempont volt a letisztultság, az egyszerűség és az egyértelműség.

A weboldal megnyitásakor a felhasználónak egy kezdőlap jelenik meg, amelynek a tetején egy navigációs sáv található. Ezen a navigációs sávon keresztül érhetőek el az oldal különböző tartalmai, továbbá itt található a keresési mező is. A keresési mező mellett egy ikon látható, amelyre kattintva egy legördülő menü nyílik ki, amely a felhasználóra vonatkozó funkcionalitásokat tartalmazza. Egy vendégfelhasználó esetén ez a legördülő menü tartalmazza a bejelentkezéshez szükséges űrlapot, valamint felkínálja a lehetőséget a Facebook API-val való bejelentkezésre, elfelejtett jelszó megoldására és regisztrálásra is (lásd 9 ábra). Egy bejelentkezett felhasználó esetén a legördülő lista tartalma megváltozik, így azután ez olyan tartalmakra való navigálást biztosít, mint például saját dalok illetve gyűjtemények megtekintése, profiladatok megváltoztatása, továbbá itt van lehetőség a kijelentkezésre is.

A számos funkcionalitás végeredménye az akkordmenetek egy részhalmazának vagy teljes összességének a kilistázása, ilyen például a keresés tartalmának a megtekintése, saját akkordmenetek megtekintése vagy egy gyűjtemény tartalmának a listázása. Az oldal az akkordmenetek



9. ábra. A bejelentkezést, a regisztrációt, valamint az elfelejtett jelszó lecserélését lehetővé tevő ablak



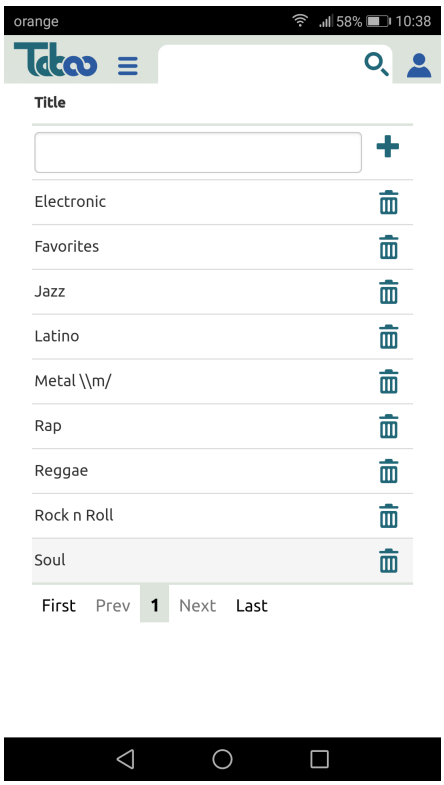
10. ábra. Az addkordmenet mobil nézetben való megjelenítése

11. ábra. Az akkordmenetek és a rajtuk végrehajtható műveletek listája oldalanként

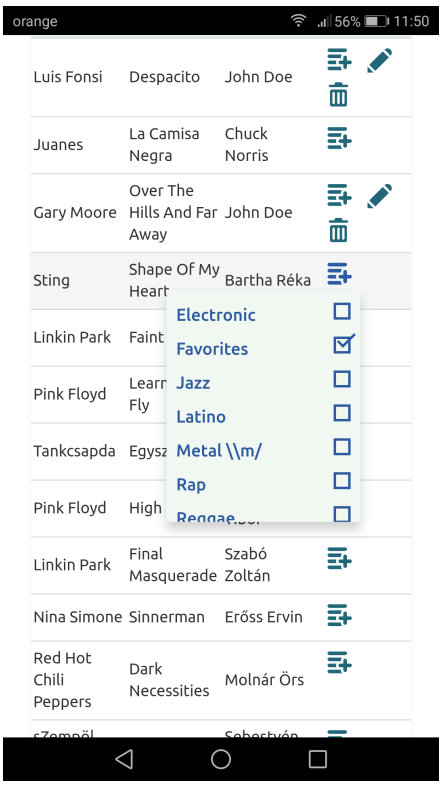
kilistázására ugyanazt a komponenst használja, így biztosít azoknak egységességet, és könnyű karbantarthatóságot. Egy ilyen listázás látható a 11. ábrán, ahol a lista sorai egy-egy akkordmenetet reprezentálnak, amelyben látható a dal szerzője, címe illetve az akkordmenet szerkesztőjének a neve. Ezek mellett a bejelentkezett felhasználó számára különböző eseménygombok jelennek, meg mint például a gyűjteménybe való rendezés, saját akkordmenet esetén a törlés illetve a szerkesztés. Az adminisztrátor illetve moderátor szerepköröknél a törlés mindegyik dal mellett megtalálható, mivel nekik jogosultságuk van ennek a műveletnek a végrehajtására.

A kilistázott elem soraira kattintva megtekinthetővé válnak az akkordmenetek (lásd 10. ábra). Az alkalmazás célkitűzései között szerepel, hogy használható legyen kisebb képernyővel rendelkező eszközökön is, ezért az oldal kialakításában fontos szerepet kap az egyszerűség és az átláthatóság. Ennek érdekében az akkordmenet megtekintésénél a dalszöveg mellett csak néhány funkciógomb került feltüntetésre, amelyet a felhasználó az akkordmenetre alkalmazhat (például a transzponálás, letöltés, szerkesztés, törlés, jelentés és gyűjteményhez való hozzáadás). Itt látható, hogy a mobilnézetben a navigációs sáv alkalmazkodik a képernyő méretéhez, így az itt elérhető menüpontok összetömörülnek, ezáltal is támogatva az alkalmazás kényelmes használatát.

Az oldal hasznos funkcionalitásai közé tartozik, hogy az akkordmenetek gyűjteményekbe rendezhetőek. A 12. ábra mutatja, hogy a felhasználó hogyan készíthet magámnak gyűjteményt. Első lépésben létrehozza azt, majd ahogyan a 13-as ábra is mutatja, a dalokat egyszerűen hozzárendeli a kívánt gyűjteményhez

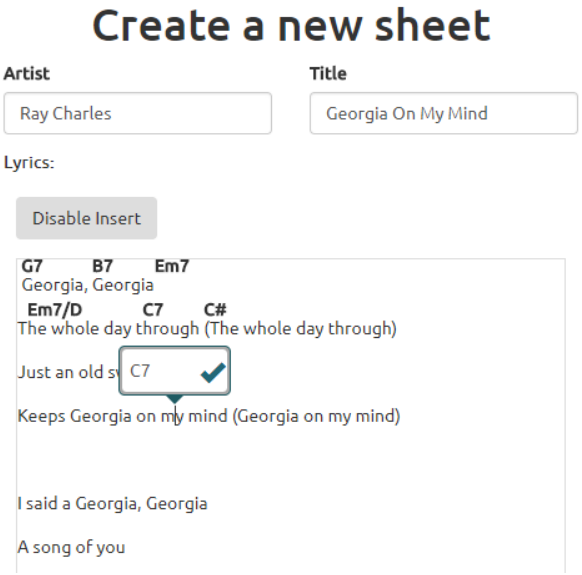


12. ábra. Az felhasználó gyűjteményeinek a listája



13. ábra. Akkordmenet gyűjteményhez való hozzáadása.

Az egyik legfontosabb eleme az alkalmazásnak az akkordmenet szerkesztő. Ennek a használata látható a 14. ábrán. A felhasználó a dalszöveg beírása után a felugró ablakba beírhatja a kívánt akkordot, majd ennek a véglegesítésével az megjelenik az akkordmenet fölött. A felhasználó valós időben látja az eredményt, ennek következtében lehetősége van az akkordmenet pontos megszerkeztésére.



14. ábra. Új akkordmenet létrehozását lehetővé tevő szerkesztő

Következtetések és továbbfejlesztési lehetőségek

A Taboo projekt keretein belül sikerült célkitűzéseinek megfelelően, egy olyan alkalmazást megvalósítani, amely az akkordmenetek szerkesztésére és kezelésére egy interaktív és felhasználóbarát felületet biztosít.

A létrehozott szerkesztő használatával, felhasználói akár néhány egérművelet segítségével megszerkeszthetik kedvenc dalaik pontos akkordmeneteit. Sikerült kiküszöbölni a sortördelés problémáját, valamint kiépíteni a reszponzív designt. Így, ezeknek köszönhetően mind az alkalmazás, mind az akkordmenetek olvasása, a kisebb készülékeken is élvezhető marad.

Emellett az alkalmazásban sor került olyan sajátosságok megvalósítására, mint például az akkordmenetek transzponálása, jelentése, PDF formátumban való letöltése vagy gyűjteményekbe rendezése. Az összeválogatot dalok, bárkivel megoszthatóak az URL által, vagy akár letölthetőek PDF formátumban. Megemlítendő, hogy sikerült kialakítani egy adminisztrációs felületet is. Itt a jelentett akkordmenetek listázhatóak, ezekről a jelentések törölhetőek, de lehetőség van akár az egész akkordmenet törlésére is.

Az alkalmazás támogatja a megfelelő felhasználó kezelést. Azok tehát belépést nyerhetnek a rendszerbe úgy regisztrációval, mint szociális háló (Facebook) használatával és ezenfelül elfelejtett jelszó esetén újat állíthatnak be.

A projekt tervezése és fejlesztése során számos új ötlet került szóba, mint továbbfejlesztési lehetőség. Néhány fontosabb funkcionalitás ezek közül:

- nemzetköziesítés;
- a keresés bővítése különböző szűrésekkel;
- a szerkesztő továbbfejlesztése (például borítókép és YouTube videó hozzárendelési lehetőségének kialakítása egy dalhoz);
- lehetőség teremtése a felhasználók (akkordmenetekre vonatkozó) javaslatainak egymás közötti megosztásásra;
- értesítésrendszer kialakítása a javaslatok számára;
- az akkordok grafikus megjelenítése az akkordmenetek felett;
- egy Chordify-hoz hasonló akkordmenet-generáló integrálása a rendszerbe.

Hivatkozások

[1] *Apache Tomcat® hivatalos weboldal*. URL: http://tomcat.apache.org/.

[2] *API documentation for creating PDF solutions | iText Developers*. URL: https://developers.itextpdf.com/apis.

[3] Alex Ben et al. *Spring Security Reference*. URL: https://docs.spring.io/spring-security/site/docs/4.2.5.RELEASE/reference/htmlsingle/#what-is-acegi-security.

[4] *Bootstrap hivatalos weboldal*. URL: https://getbootstrap.com/ (utolsó elérés dátuma: 2018. márc. 26.)

[5] *Browserify hivatalos weboldal*. URL: http://browserify.org/.

[6] *Checkstyle hivatalos weboldal*. URL: http://checkstyle.sourceforge.net/.

[7] Walls Craig et al. *Spring Social Reference*. URL: https://docs.spring.io/spring-social/docs/1.1.4.RELEASE/reference/htmlsingle/.

[8] *Docker Documentation*. URL: https://docs.docker.com/engine/reference/builder/#usage.

[9] *Draft.js hivatalos weboldal*. URL: https://draftjs.org/ (utolsó elérés dátuma: 2018. márc. 26.)

[10] *FindBugs™ hivatalos weboldal*. URL: http://findbugs.sourceforge.net/.

[11] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. URL: https://martinfowler.com/articles/injection.html.

[12] *FreeMarker Java Template Engine*. URL: https://freemarker.apache.org/.

[13] Oliver Gierke, Thomas Darimont, és Christoph Strobl. „Spring Data JPA -Reference Documentation version”. (). URL: https://docs.spring.io/spring-data/data-jpa/docs/1.6.0.RELEASE/reference/pdf/spring-data-jpa-reference.pdf.

[14] *Gradle Build Tool hivatalos weboldal*. URL: https://gradle.org/.

[15] *gulp.js hivatalos weboldal*. URL: https://gulpjs.com/.

[16] *H2 hivatalos weboldal*. URL: http://www.h2database.com/html/main.html.

[17] *Hibernate ORM hivatalos weboldal*. URL: http://hibernate.org/orm/.

[18] *Immutable.js hivatalos weboldal*. URL: https://facebook.github.io/immutable-js/docs/#/.

[19] *JavaMail Reference Implementation*. URL: https://javaee.github.io/javamail/.

[20] Cody Lindley. *React Enlightenment*. URL: https://www.reactenlightenment.com/ (utolsó elérés dátuma: 2018. márc. 26.)

[21] *MariaDB hivatalos weboldal*. URL: https://mariadb.org/.

[22] Webb Phillip et al. *Spring Boot Reference Guide*. URL: https://docs.spring.io/spring-boot/docs/1.5.4.RELEASE/reference/htmlsingle/.

[23] *Project Lombok hivatalos weboldal*. URL: https://projectlombok.org/.

[24] *React hivatalos weboldal*. URL: http://www.reactjs.org/ (utolsó elérés dátuma: 2018. márc. 26.)

[25] *Redux hivatalos weboldal*. URL: https://redux.js.org/ (utolsó elérés dátuma: 2018. márc. 26.)

[26] Johnson Rod et al. *Spring Framework Reference Documentation*. URL: https://docs.spring.io/spring/docs/4.3.9.RELEASE/spring-framework-reference/htmlsingle.

[27] *The Flying Saucer User's Guide*. URL: https://flyingsaucerproject.github.io/flyingsaucer/r8/guide/users-guide-R8.html#xil_3.

[28] *TSLint hivatalos weboldal*. URL: https://palantir.github.io/tslint/.

[29] *TypeScript hivatalos weboldal*. URL: http://www.typescriptlang.org/ (utolsó elérés dátuma: 2018. márc. 26.)

[30] *Yarn hivatalos weboldal*. URL: https://yarnpkg.com/lang/en/.