EÖTVÖS LORÁND TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR

Ütőhangszerek eredetének és elterjedésének interaktív, multimédiás webtérképes ábrázolása

SZAKDOLGOZAT

FÖLDTUDOMÁNYI ALAPSZAK

TÉRKÉPÉSZ ÉS GEOINFORMATIKA SPECIALIZÁCIÓ

Készítette:

Kun Barka

Témavezető:

Dr. Gede Mátyás

Egyetemi docens

Külső témavezető:

Holló Miklós

ELTE Térképtudományi és Geoinformatikai Intézet

Budapest, 2023

Nyilatkozat

Alulírott, Kun Barka nyilatkozom, hogy jelen szakdolgozatom teljes egészében saját, önálló szellemi termékem. A szakdolgozatot sem részben, sem egészében semmilyen más felsőfokú oktatási vagy egyéb intézménybe nem nyújtottam be. A szakdolgozatomban felhasznált, szerzői joggal védett anyagokra vonatkozó engedély a mellékletben megtalálható.

A témavezető által benyújtásra elfogadott szakdolgozat PDF formátumban való elektronikus publikálásához a tanszéki honlapon

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, 2023. május 15.

Kun Barka

A hallgató aláírása

Tartalomjegyzék

Bevezetés

Szakdolgozatomban a tradicionális és modern ütőhangszerek egy részének területi elterjedését és adott esetben származási helyét szeretném feltüntetni interaktív, webtérképes bemutatásban.

Ismeretterjesztő és oktatási céllal gondoltam összevonni geoinformatikai és hangszeres tanulmányaimat, és az előbbit kihasználva információt szolgáltatni az ütőhangszerek sokféleségéről, használatáról, elterjedéséről. Az online környezet miatt egyszerűen elérhető mindenki számára, a vizuális ábrázolás pedig könnyíti az elképzelést, az információk befogadását, összekapcsolását.

Mindig érdekesnek találtam a tradicionális hangszerek, használati tárgyak „vándorlását", és hogy ezek hogyan alakultak különböző környezetekben, kultúrákban, ahová eljutottak akár felfedezők, akár egyéb utazások hatására, illetve azt, hogy milyen területeken alakultak ki nagyon hasonló hangszerek egymástól teljesen függetlenül. Ezeket a kapcsolatokat, vagy épp azok hiányában is létező hasonlóságokat hivatott összefoglalni a munkám, amit, mint említettem, akár oktatási céllal is fel lehet majd használni a szabad hozzáférés, interaktivitás, vizualizációk és multimédiás feldolgozás miatt.

A cél az ütőhangszerek elterjedésének és származási helyének interaktív webtérképes vagy földgömbös ábrázolása volt. Végül az elterjedési felületet jelenítettem meg, a származási helyet információként közöltem. Az eredeti elképzelés egy Google Earth jellegű megjelenítés volt, ahol, ha rávisszük a kurzort, megjelennek a poligonok[1], [2], az információk pedig az ikonokra való kattintáskor tűnnek fel. Ezen időközben változtattam, mivel sík térképen jobban átlátható a Föld egészén az elterjedés. Így a háttértérkép Google Maps lett, ezen jelennek meg az adott információk, geometriák.

Felhasználóbarát felületet szerettem volna létrehozni, így terv szerint van rajta keresési lehetőség, illetve oldalt egy lista a hangszerekről.

A projekten az alábbi linken érhető el: www.kunbarkapr.hu

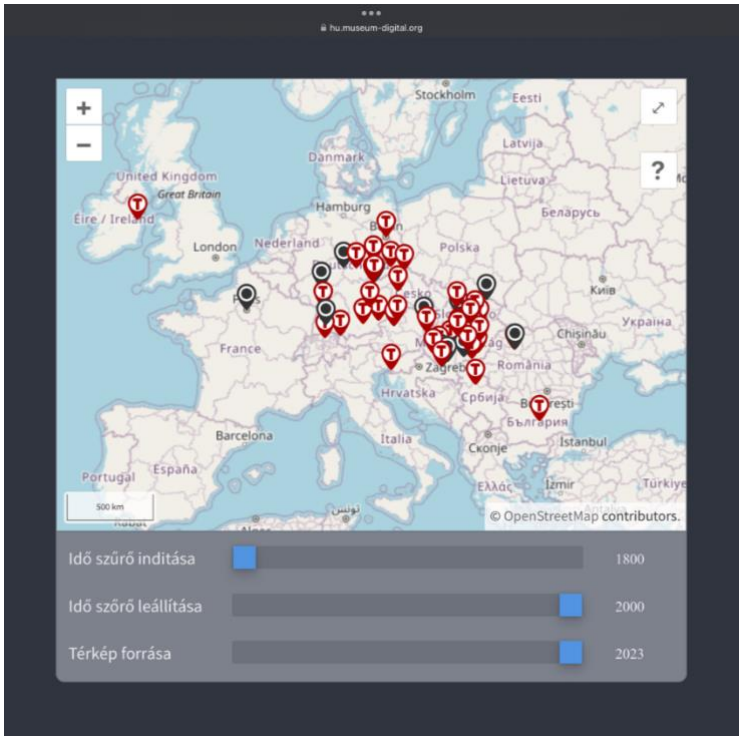
Történeti, irodalmi áttekintés, kapcsolódó korábbi témák

Az ütőhangszerekről

Az ütőhangszer a legősibb hangszertípus: már két fadarab összeütésével is létre tudunk hozni hangokat. Területenként másképpen fejlődtek, alakultak, ez főként attól függött, hogy milyen anyagok álltak rendelkezésre, mit tudtak használni a hangszerek kialakítására, illetve milyen módszerekkel készítették azokat. Ennek ellenére sok helyen alakultak ki nagyon hasonló hangszerek, például a timpani, vagyis az üstdob használatos Egyiptomban, Görögországban, illetve a hébereknél is alakult ki hasonló hangszer. Sok esetben nehéz megállapítani, hogy a vándorló törzsek magukkal vittek-e adott hangszereket, és azok ezért terjedtek el, vagy egymástól függetlenül alakultak ki. Használatuk módja is többféle lehetett: egyszerűen csak zenei célokra készítették őket, esetleg üzenetküldési funkciójuk, illetve rituális alkalmazásuk is volt. Afrikai és egyéb törzseknél igen gyakori a zene, illetve a hozzá kötött táncok, „énekek" rituális, vallási jelentősége. Máshol a sámánok személye van/volt összekötve ütőhangszerrel (pl. sámándob), és az általa létrehozható spirituális kapcsolatokkal.

Korábbi témák

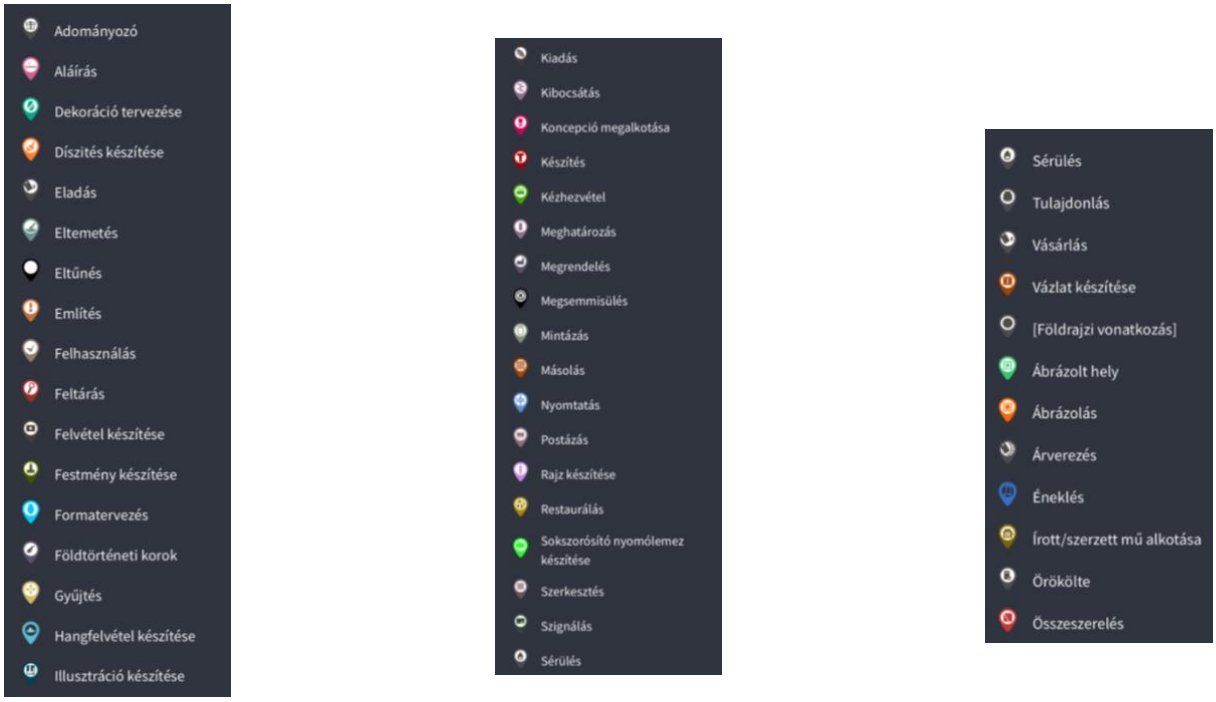
Hasonló, kapcsolódó témák keresése során tapasztaltam, hogy különböző gyűjtemények kaptak többféle megjelenítést, többek között térképit is. A hangszeres vonatkozáshoz a Múzeum Digitár oldalán a Zenetörténeti Múzeum gyűjteménye kapcsolódik, amelynek címe *Falusi, városi, nemzeti és egzotikus hangszerek gyűjteménye*.[3]



1. ábra – A Múzeum Digitár térképes ábrázolása

Háttértérképnek az OpenStreetMapet[4], és a historical-basemaps adatkészletet[5] használták, az interakciókra egy időszűrő 2 csúszkával (eleje, vége években) és a háttértérkép dátumbeállítója áll rendelkezésre, így a háttértérkép megjelenése és a feltüntetett pontszerű jelek száma is változtatható.

Térképészeti szempontból negatívumként szeretném megemlíteni, hogy a jelmagyarázatban mindig, minden térképi megjelenítésüknél benne van az összes jel, nem csak a jelen esetben használt 2, ezért inkább jelkulcsnak nevezném. Az adott gyűjteménynél így nehézkesebb megtalálni, hogy a sok jel közül éppen melyek az aktuálisak.

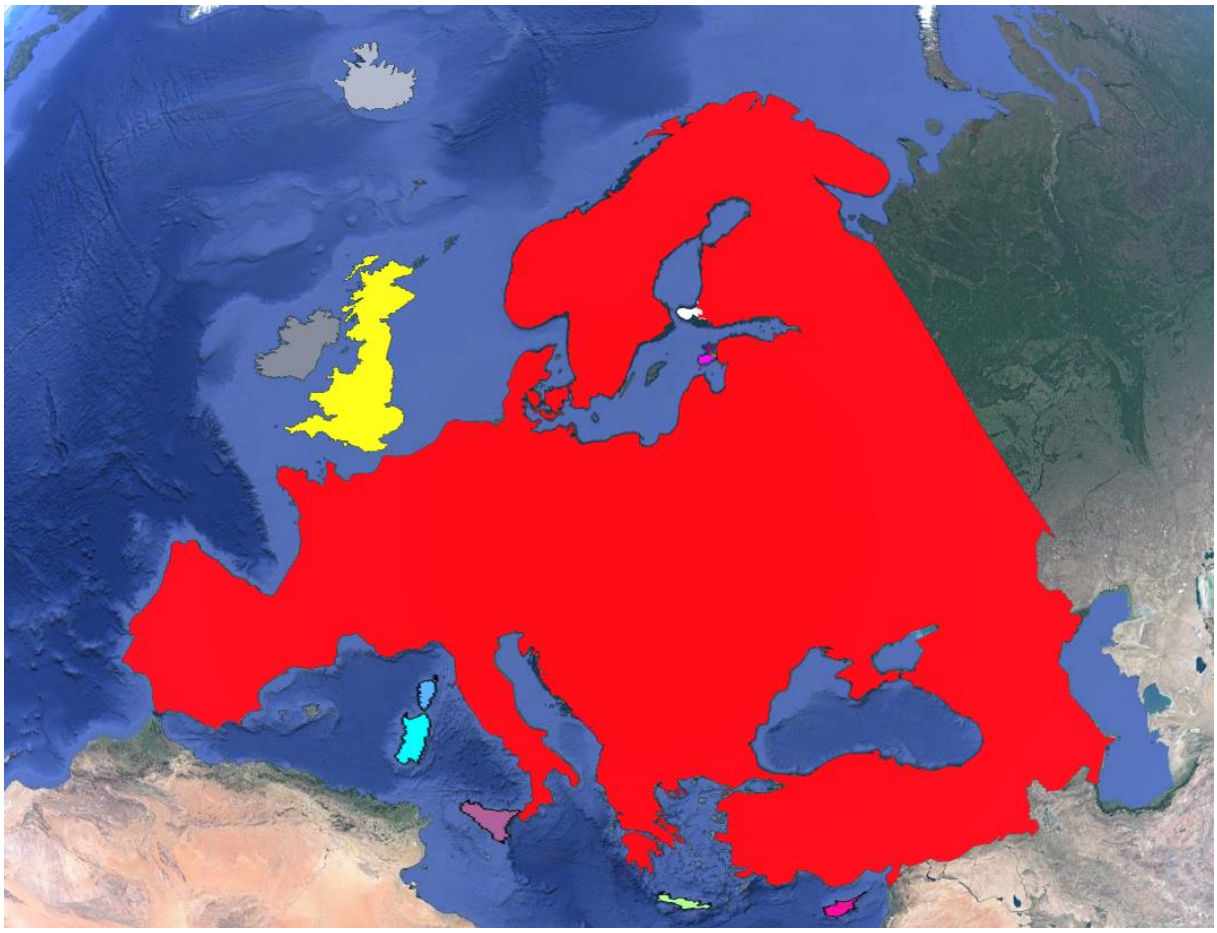


2. ábra – Jelkulcs

Módszertan

Térképészeti módszerek

Miután utánanéztem, hogy mely hangszereket hol használják, hol terjedtek el, poligonokat rajzoltam, mert ezt az ábrázolási módszert találtam a legmegfelelőbbnek: mivel körvonala és kitöltése is van, jól látható az általa fedett felület. Így poligonok kerültek az adott területekre, multipoligonként. Ennek lényege, hogy több, egymástól független poligont egy elemként lehet kezelni, például az Európa multipoligon több poligonból áll, a kontinens, illetve a hozzá tartozó szigetek így megjeleníthetőek egy elemként, egy Európát alkotva. A poligonokat KML (Keyhole Markup Language) [6] formátumban mentettem. Azért választottam ezt az XML-re (Extensible Markup Language)[7] épülő formátumot, hogy PHP[8]-ban könnyebben dolgozhassak az adatokkal, mert az információ struktúráját írja le, és földrajzi annotációkra specifikus. Így a KML-nek a paramétert tartalmazó tulajdonsága, illetve az, hogy alakzatokat adhatunk hozzá ebben a formátumban, segítette az importálási folyamatot. Az adat ezután adatbázisba menthető, utolsó lépéskén pedig megjeleníthető Google Maps-en.



3. ábra – az Európára vonatkozó poligonok külön-külön megnyitva Google Earth-ben

A poligonokon belül egy pontszerű elem, pointer vonatkozik egy-egy hangszerre, amivel megjeleníthető a hangszer poligonja és a hozzá tartozó információk, adatok a hozzárendelt infoboxban.

4. ábra – 2 pointer Európára vonatkozó multipoligonhoz a projekt korai szakaszában

Fejlesztési módszer

Mivel online történik a felület megjelenítése, különös figyelmet kapott a betöltési sebesség, illetve a modern böngészők teljes mértékű támogatása.

Ezek miatt Laravel[9], [10] backend keretrendszerben készült a projekt, ami kiemelkedő teljesítményt nyújtó, sok funkciót kínáló, illetve skálázható keretrendszer. Ezek mellett jelentős előnye az ORM-je (Object-relational mapping)[11], mivel ennek köszönhetően érhető el részben a gyors betöltési sebesség, valamint megkímélt a Nativ SQL (Structured Query Language)[12] query-k írásától is, ezzel is gyorsítva és egyszerűsítve a fejlesztési folyamatot. A Model View Controller (MVC)[13] elvet követi, ezért hatékonyabb és biztonságosabb, mint a natív PHP. Megkönnyíti az olyan általános feladatok elvégzését, mint a hitelesítés (authentication), a munkafázis (sessions), a routing, vagy a gyorsítótárba másolás (caching). Egyedülálló felépítéssel rendelkezik, melynek segítségével lehetővé válik, hogy egy, a specifikusan az alkalmazásunkhoz igazított, saját infrastruktúrát hozzunk létre.

A projekt megvalósítása közben különös figyelmet fordítottam a weboldal reszponzivitására, ehhez két lehetőség merült fel: a TailwindCSS, illetve a Bootstrap[14], és végül az utóbbi mellett döntöttem a fejlesztőbarátabb dokumentációja, valamint működési elve miatt is.

Térképi megjelenítés szempontjából három megoldás jött szóba, a Google Maps, a Leaflet és az OpenStreetMap, végül a Google Maps lett a kiválasztott, részben a részletes dokumentációja, illetve a fejlesztőbarát felépítése miatt, másrészt ez nyújtja a leggördülékenyebb felhasználói

élményt is, emellett gyors, és rendkívül jól kezeli, ha több tíz-száz-ezer koordinátából alkotunk poligonokat egyszerre, valamint rendelkezik ingyenes teszthozzáféréssel, ami elengedhetetlen szempont volt a fejlesztés során.

A webszerverek közül a Cweb[15] szolgáltató mellett döntöttem, mert kiváló ár-érték arányt biztosítanak, emellett az ország 12 pontján napi biztonsági mentést végeznek díjtalanul, automatikusan a szerver állapotáról.

Általános tudnivalók

Általános leírás, verziószámok

A projekt frontend tekintetében HTML5, CSS3, Javascript kódolású, backend tekintetében Laravel keretrendszert használ, illetve MySQL[16] adatbázison alapszik. Az 5. ábrán látható a felépítése. A reszponzív kezelés Bootstrap, illetve CSS media querykkel valósul meg.

PHP: v8.2.1

Composer[17]: v2.5.1

Laravel: v9.48.0

Bootstrap: v5.3.0

MySQL: v10.4.11-MariaDB

Adatbázis felépítése, szerkezete

Az adatbázis[18] szerkezete

Instruments: Hangszer adatainak tárolása.

Instrument_types: A hangszerek típusai.

Instruments_locations: A hangszerek és a hozzájuk tartozó poligonok közötti kapcsolótábla.

Locations: A poligon nevét tárolja.

Coordinates: A poligonokhoz tartozó koordináták tárolása.

5. ábra - Az adatbázis felépítése

Webes felület kialakítása

Laravel általános leírása

Laravel

A Laravelből 4 dolgot emelnék ki részletesen, ami segítségemre volt. A migrációk, hogy elkerüljem az SQL queryket, a modellek, hogy megfeleljek az MVC (Model-View-Controller) elveknek, illetve a kód olvashatósága érdekében a Controllerek, ahol a logikai réteg helyezkedik el, és a Blade, ami a megjelenítést segíti.

Migrációk

A migrációk a Laravel alapvető funkciói, amelyek lehetővé teszik az adatbázis sémájának kezelését. Az adatbázisok struktúrájának változása az idő múlásával gyakori, és a migrációk segítenek az ilyen változások kezelésében.

A Laravel migrációk segítségével lehetőségünk van az adatbázis sémájának létrehozására, módosítására vagy törlésére. Ezek a fájlok általában a database/migrations mappában találhatók.

A migrációk létrehozása után ezek a Laravel Artisan parancssorának segítségével futtathatók. Az Artisan migrációs parancsai közé tartozik például a migrációk létrehozása (php artisan make:migration), a migrációk futtatása (php artisan migrate) vagy a migrációk visszavonása (php artisan migrate:rollback).

A migrációk használatával könnyen és biztonságosan kezelhetjük az adatbázis sémájának változásait a Laravel alkalmazásban.

Modellek

A modellek a Laravelben az MVC tervezési mintában az alkalmazás modell rétegének részét képezik, és az adatbázissal való kommunikációért felelősek.

A modellek létrehozásához csak egy egyszerű PHP osztályt kell definiálni, amely összekapcsolja az adatbázis-táblákat és az alkalmazásunkat.

A modellek segítségével könnyedén tudunk például adatokat beszúrni, módosítani vagy törölni az adatbázisból, és ezek az adatok szorosan összekapcsolódnak az alkalmazásunkkal.

A Laravelben a modellek használata a hozzáférési réteg egyszerűsítésére és az adatbázissal való kommunikáció elválasztására szolgál. Ezek az osztályok lehetővé teszik, hogy a kódunk

áttekinthetőbb legyen, és az adatbázisműveletekkel kapcsolatos kódrészletek egy helyen legyenek, így egyszerűbb és biztonságosabb lesz az alkalmazás karbantartása és bővítése.

Controllerek

A controllerek a Laravel MVC tervezési mintájában az alkalmazás vezérlő rétegének részét képezik. Feladatuk, hogy kezeljék az alkalmazás kéréseit (request) és visszaadják a megfelelő választ (response).

A controllerek felelősek az adatok feldolgozásáért, valamint a válasz visszaadásáért az alkalmazás felhasználói felületén. Az adatok feldolgozását az általuk használt modellekkel végzik, majd az eredményt visszaküldik a megfelelő nézetnek.

A controllerek legfontosabb feladatai közé tartozik a kérések kezelése, a felhasználói bemenet validálása, az adatok feldolgozása és a válasz visszaküldése. A controllereknek lehetnek beépített metódusai, amelyek segítségével elérhetjük az adatbázist és az alkalmazás más részeit, valamint saját metódusokat is definiálhatunk az adott feladat végrehajtásához.

A Laravelben a controllerek használata szinte elengedhetetlen az alkalmazások fejlesztéséhez. A controllerek létrehozása is nagyon egyszerű, és a beépített funkcionalitásnak köszönhetően nagyban megkönnyítik az alkalmazásfejlesztést.

Blade

A Blade egy szerveroldali sablonmotor, melyet a Laravel keretrendszerben használnak. A sablonmotor lehetővé teszi a fejlesztők számára, hogy az alkalmazások felhasználói felületét könnyen és hatékonyan létrehozzák. A Blade sablonmotor az egyszerűségre és a hatékonyságra összpontosít, így lehetővé teszi, hogy a fejlesztők gyorsabban és hatékonyabban hozzanak létre sablonokat.

A Laravel Blade sablonmotorjának előnyei:

1. Összhangban van a PHP szintaxisával, így a Laravel Blade sablonmotorjával való munka egyszerű és intuitív.
2. Tartalmaz olyan funkciókat, mint például a feltételes logikai ágak és ciklusok, amelyek lehetővé teszik a kód hatékonyabb és olvashatóbb írását.
3. A sablonmotor a sablonokat előre fordítja, így a sablonok futási sebessége gyorsabb, mint más sablonmotoroké.
4. A sablonmotor lehetővé teszi a sablonok komponensekre bontását, melyek újrahasznosíthatók az alkalmazás különböző részeiben.

Összességében a Laravel Blade sablonmotorja lehetővé teszi a fejlesztők számára, hogy gyorsan és hatékonyan hozzanak létre dinamikus, átlátható és karbantartható felhasználói felületeket az alkalmazásuk számára**.**

Laravel részei kód szinten

Migrációk:

A migrációknak csak az *up* metódusát emelem ki, ahol látszanak a kapcsolatok, illetve hogy milyen mezőket tartalmaz.[19]

```
public function up()
{
    Schema::create('instruments', function (Blueprint $table) {
        $table->id()->autoIncrement();
        $table->foreignId('instrument_type_id');
        $table->string('name')->nullable();
        $table->string('origin_name')->nullable();
        $table->string('origin')->nullable();
        $table->text('description')->nullable();
        $table->text('example')->nullable();
        $table->boolean('has_polygon')->default(true);
        $table->string('color', 7);
        $table->string('pointer_latitude');
        $table->string('pointer_longitude');
        $table->text('youtube_url')->nullable();
        $table->text('sound_url')->nullable();
        $table->text('image_url')->nullable();
    });

    Schema::table('instruments', function (Blueprint $table) {
        $table->foreign('instrument_type_id')
            ->references('id')
            ->on('instrument_types');
    });
}
```

```
public function up()
{
    Schema::create('instrument_types', function (Blueprint $table) {
        $table->id();
        $table->string('name');
    });
}
```

```
public function up()
{
```

```
    Schema::create('locations', function (Blueprint $table) {
        $table->id();
        $table->string('name');
    });
}
public function up()
{
    Schema::create('instruments_locations', function (Blueprint $table) {
        $table->id();
        $table->unsignedBiginteger('instruments_id')
            ->unsigned();
        $table->unsignedBiginteger('locations_id')
            ->unsigned();
        $table->foreign('instruments_id')
            ->references('id')
            ->on('instruments')
            ->onDelete('cascade');
        $table->foreign('locations_id')
            ->references('id')
            ->on('locations')
            ->onDelete('cascade');
    });
}


public function up()
{
    Schema::create('coordinates', function (Blueprint $table) {
        $table->id();
        $table->foreignId('locations_id');
        $table->string('latitude');
        $table->string('longitude');
    });

    Schema::table('coordinates', function (Blueprint $table) {
        $table->foreign('location_id')
            ->references('id')
            ->on('locations');
    });
}
```

Modellek:

A modellek közül csak azokat emelem ki, amelyekbe került egyedi metódus is, mivel a migrációknál látható, hogy melyikhez milyen értékek tartoznak.

Locations model:

Instruments metódus -> visszaadja a helyhez tartozó hangszereket

coordinates metódus -> visszaadja a helyhez tartozó koordinátákat

Instruments model:

Type metódus -> visszaadja a hangszer típusát

locations metódus -> visszaadja a hangszerhez tartozó poligonokat

Coordinate model:

Lat és *Lng* metódusok, két egyedi attribútumot hoztam létre, hogy a szélességi (latitude) és hosszúsági (longitude) értékeket lat és lng formátumban kapjam meg, így szebben adható át a Google Mapsnek, és nem szükséges kézzel átnevezni egy ciklusban sem.

Egyben a legérdekesebb modell bemutatása:

```
class Coordinate extends Model
{
    use HasFactory;
    public $timestamps = false;
    protected $visible = ['lat', 'lng'];
    protected $appends = ['lat', 'lng'];
    public function Lat():Attribute{
        return Attribute::make(
            get: fn () => doubleval($this->latitude)
        );
    }
    public function Lng():Attribute{
        return Attribute::make(
            get: fn () => doubleval($this->longitude)
        );
    }
}
```

Controllerek:
ViewController

```
public function showMap(): Factory|View|Application
{
    $instrumentsFormattedInJson = array();
    $instruments = Instruments::with(
        ['locations', 'locations.coordinates']
    )->get();
    foreach ($instruments as $instrument){
        $polygons = array();

        if ($instrument->has_polygon){
            $locations = $instrument->locations;
            foreach ($locations as $location){
                $polygons[] = $location->coordinates;
            }
        }
```

```
        $instrumentsFormattedInJson [] = array(
            'id' => $instrument->id,
            'type_id' => $instrument->type->id,
            'type_name' => $instrument->type->name,
            'name' => $instrument->name,
            'origin_name' => $instrument->origin_name,
            'origin' => $instrument->origin,
            'description' => $instrument->description,
            'example' => $instrument->example,
            'has_polygon' => $instrument->has_polygon,
            'color' => $instrument->color,
            'pointer_latitude' =>
                doubleval($instrument->pointer_latitude),
            'pointer_longitude' =>
                doubleval($instrument->pointer_longitude),
            'youtube_url' => $instrument->youtube_url,
            'sound_url' => $instrument->sound_url,
            'image_url' =>
                $instrument->image_url ?
                    asset($instrument->image_url) : null,
            'polygons' => $polygons,
        );
    }

    return view('welcome', [
        'instruments' => $instrumentsFormattedInJson
    ]);
}
```

Ez a forráskódrészlet először lekéri az Instruments modelleket az azokhoz tartozó helyszínekkel és koordinátáikkal együtt, majd végigmegy a lekért Instruments modelleken, és formázza azokat egy tömbben, ezután visszatér a welcome nézetben megjelenítendő formázott adatokkal.

A $instrumentsFormattedInJson egy üres tömb, amiben az összes formázott Instruments modellt tároljuk. Az Instruments::with(['locations', 'locations.coordinates'])->get() kódrész lekérdezi az összes Instruments modellt az azokhoz tartozó locations modellekkel és azok koordinátáival együtt. Az Instruments modellek és az azokhoz tartozó kapcsolódó locations modellek a Laravel Eloquent[20] ORM segítségével vannak definiálva.
A foreach ciklusban végigmegyünk az összes Instruments modellen, és ha az adott Instruments modellnek van poligonja (has_polygon), akkor a hozzá tartozó locations modellek koordinátáit eltároljuk a polygons tömbben. Ha az adott Instruments modellnek nincs poligonja, a polygons tömb üres marad.

A kód végül egy asszociatív tömböt készít az összes formázott Instruments modellből, majd visszatér a welcome nézettel, amelynek átadja az összes formázott Instruments modellt a $instruments változóban. A welcome nézetnek az adatok megjelenítését kell végrehajtania.

Api/ImporterController:

```
public function importKmlToExistingLocation(Request $request){

    set_time_limit(300);
    ini_set('max_execution_time', 300);
    $kmlArray = $this->kmlToArray($request->kmlFile);

    $location = Locations::findOrFail($request->locationId);

    $coordinatesToDelete =
        Coordinate::all()
            ->where('locations_id', '='. $location->id);
    foreach ($coordinatesToDelete as $coordinateToDelete){
        $coordinateToDelete->delete();
    }

    echo 'Deleted completed';

    foreach ($kmlArray as $key => $kmlLocationCoordinate){
        $newCoordinate = new Coordinate();
        $newCoordinate->latitude = $kmlLocationCoordinate[1];
        $newCoordinate->longitude = $kmlLocationCoordinate[0];
        $newCoordinate->locations_id = $location->id;
        $newCoordinate->save();
    }

    return response('Import was successfully', 200)
        ->header('Content-Type', 'text/plain');
}
```

Ez a funkció az adatbázisba történő KML importálását végzi, amely egy POST kérést kezel, és KML adatokat importál egy meglévő helyszínbe. Innen már az insomnia[21] Api teszterrel vannak importálva az adatok.

Működése:

1. A kód beállítja az időkorlátot erre a szkriptre 5 percre (300 másodperc) és a maximális végrehajtási időt is 5 percre.
2. A KML fájlt tömbbé alakítja a kmlToArray függvény segítségével, amely nem szerepel ebben a kódban.
3. A helyszín azonosítójával lekéri a helyet az adatbázisból.

4. A helyhez tartozó összes koordinátát törli az adatbázisból.
5. Egy ciklus fut végig a KML koordinátáinak tömbjén. Minden koordinátához létrehoz egy új Coordinate modelt, és beállítja annak szélességi és hosszúsági értékét és a megfelelő helyszín azonosítóját.
6. A kódban lévő új koordinátát elmenti az adatbázisba.
7. Sikeres mentés esetén egy üzenetet ír ki a képernyőre.
8. Egy 200-as státuszkóddal és a text/plain tartalomtípussal egy választ ad vissza.

A 2-es pontban említett kmlToArray funkció:

```
private function kmlToArray($filePath): array|string
{
    if (file_exists($filePath)) {
        $xml = simplexml_load_file($filePath);
        $placeMarks = $xml->Document->Placemark[1];
            $placeMark = (string)$placeMarks
                ->Polygon->outerBoundaryIs
                ->LinearRing->coordinates;
            $values = explode(" ", trim($placeMark));
            $coords = array();
            foreach ($values as $coord) {
                $value2 =
                    trim(preg_replace(
                        '/\t+/',
                        '',
                        $coord));
                $args = explode(",", $value2);
                $coords[] = array($args[0], $args[1]);
            }
    } else {
        return 'failed_to_open_file';
    }
    return $coords;
}
```

Egy PHP kódrészlet, ami az adott elérési úton található fájlból betölt egy XML fájlt, ezután a koordinátákat kinyeri az XML fájlból, és eltárolja őket egy koordináta tömbjében.
Az if utasítás ellenőrzi, hogy az adott fájl létezik-e a megadott elérési úton, ha igen, akkor betölti a fájlt a simplexml_load_file függvénnyel. Ez betölti az XML fájlt, és lehetővé teszi az XML adatokat tartalmazó objektumok használatát.
Ezután a kód a Placemark[1] taget találja meg a betöltött XML fájlban, majd kinyeri a koordinátákat a Polygon tagből, és azokat eltárolja egy tömbben.

Ha az adott elérési úton a fájl nem található, a kód visszatér a 'failed_to_open_file' üzenettel. Ez a kódrészlet része lehet egy nagyobb programnak, ami fájlok és adatok feldolgozására szolgál.

Blade “welcome” nézet:
Itt található az oldal szerkezete. 3 részt emelek ki. A kereső panelt, amire a későbbiekben hivatkozni fogok, a hangszer panelt, itt nyer igazán értelmet a blade használata, mert nem kell mindent javascript segitségével elrendezni, illetve a térképi adatok átadásakor hasonlóképpen.

Ez a HTML kód egy lebegő panelt hoz létre, amelynek tartalma a #instrument-floating-panel-content div-ben található. Az első div instrument-group osztályú, és egy fejlécet tartalmaz, amely a "Membranofonok" szöveget jeleníti meg. Ezután egy foreach ciklus következik, amely végigmegy az $instruments tömbön, és ha az adott elem típusa 1, akkor létrehoz egy instrument-group-item osztályú div-et, amely az elem nevét jeleníti meg a megfelelő színnel. A következő div ugyanazt a folyamatot hajtja végre az "Idiofonok" címmel.

A megfelelő szín[22] a színkódolásra utal, minden hangszer poligonja ugyanolyan színnel rendelkezik, mint a leírásban a hangszer neve, típusa, eredeti neve és származási helye, illetve oldalt a listában is, így könnyebb összekötni a hangszereket az elterjedésükkel.

Keresés szempontjából fontosnak tartottam, hogy több különböző módon elérhetőek legyenek a hangszerhez tartozó információk, így a keresőnek és a külön listának köszönhetően nem kell a pointerek között keresgélni a hangszert, amiről többet szeretnénk tudni.

- Az oldal felső részén található egy kereső funkció, itt a beírt karakterekkel a hangszer nevében és a leírásában keresünk, és a kiválasztott keresett elemre kattintva ugyanúgy megjeleníthető a hangszerhez tartozó (multi)poligon és a leírás, vagy ha nem tartozik hozzá poligon, akkor csak a leírás.
- Jobb oldalon található egy hangszerlista, ez elemenként kattintható, és megjeleníti a hangszerhez tartozó poligont, infoboxot.

Ha tudjuk, mit keresünk, a searchbar egy lehetőség, ha viszont nem tudjuk, akkor az oldalt elhelyezett lista megmutatja a választási lehetőségeket.

Frontend kialakítása

Reszponzivitás és alapinformációk a frontenden használt technológiákról:

Bootstrap

Ez egy nyílt forráskódú frontend webes keretrendszer, amivel gyors és hatékony weboldalakat, webalkalmazásokat hozhatnak létre a fejlesztők. Erősen támaszkodik a HTML, CSS és JavaScript technológiákra, illetve előre definiált stílusok és osztályok segítségével gyorsítja az oldalak fejlesztésének folyamatát.

Az oldalak tartalmát 12 oszlopra bontja, így a fejlesztők könnyebben és hatékonyabban tudják kezelni az elrendezést és az elhelyezkedést. Ez a rendszer lehetővé teszi a tartalom dinamikus elrendezését, illetve az oszlopok arányosan változnak a képernyőméretek és eszközök alapján.

A 12 oszlopos rendszerrel a Bootstrap lehetővé teszi a reszponzív design könnyű és hatékony kezelését. Az oszlopok arányainak beállításával könnyen testreszabhatóak az oldalak és alkalmazások az eszközök típusaihoz és a képrenyőméretekhez.

Ezek a dinamikus elrendezési rendszerek segítenek az oldalak és alkalmazások létrehozásában, így ezek megfelelnek a modern webes tervezési trendeknek, felhasználói elvárásoknak. Az oszlop alakú megközelítés lehetővé teszi, hogy könnyen kezelhetőek legyenek az oldalak elrendezései, tartalmai, ezzel időt takarítva meg a fejlesztőknek és maga a fejlesztés is gyorsabb és hatékonyabb lehet.

Bootstrap Modal

A Bootstrap Modal olyan interaktív megjelenítési elem, amely lehetővé teszi a felhasználók számára, hogy az aktuális oldalon egy kis felugró ablakban tartalmat jelenítsenek meg, például üzenetet, űrlapot vagy képet.

A Bootstrap Modalokat Bootstrap CSS keretrendszer használatával lehet létrehozni. Az alapvető működési elv az, hogy a Bootstrap Modalhoz tartozó HTML kódot a weboldal forráskódjába helyezzük be, majd JavaScript segítségével vezéreljük a megjelenítését és viselkedését.

Amikor egy felhasználó (például egy gombnyomással) interakcióba lép egy Bootstrap Modal hivatkozással, a Modal felugrik az oldal közepén, és eltakarja az összes többi tartalmat. A Modal lehetővé teszi a felhasználó számára a tartalom megtekintését, a beviteli mezők kitöltését és az

interakciót a Modalban található elemekkel. Ha a felhasználó befejezte az interakciót a Modalban, a Modal eltűnik, és a felhasználó a főoldalon folytatja az interakciót.

A Bootstrap Modalok nagyon rugalmasak és testreszabhatóak, lehetővé téve az előzetes beállítások, mint például a méret, a háttér, a cím és az animáció testreszabását a felhasználói igényekhez.

JavaScript

Ez egy dinamikus, interpreteres és objektumorientált programozási nyelv, ami a böngészőkben futtatva weboldalak és webalkalmazások interaktivitsát és dinamizmusát biztosítja. A JavaScript több platformon használható, ideértve a böngészőket, szerver oldali fejlesztést és akár az asztali alkalmazásokat is. Lehetővé teszi a weboldalak dinamikus tartalmának létrehozását, például a felhasználói interakciók kezelését, animációk, vizuális elemek, effektusok megjelenítését, adatok lekérdezését, azok dinamikus megjelenítését, űrlapok ellenőrzését, azokkal való munkát, valamint a szinkron adatkezelést. A JavaScript előnye, hogy böngészőben futtatva egyértelműen interaktív felhasználói élményt biztosít, mert az oldalak nemcsak statikus tartalmat jelenítenek meg, de a felhasználóval való interakciók folyamán állandóan változnak. Ez az egyik legelterjedtebb és legnépszerűbb programozási nyelv a világon, köszönhetően széles körű támogatottságának, ökoszisztémájának és kiterjedt fejlesztői közösségének[23].

```
mapElement.addEventListener("mouseup", (event) => {
        hideAllPolygons();
        setSearchPanelVisibility(true);
        closeInstrumentPanel();
});

mapElement.addEventListener("touchstart", (event) => {
        hideAllPolygons();
        setSearchPanelVisibility(true);
        closeInstrumentPanel();
});

instrumentsJson.forEach((instrument) => {
    let currentPolygon = new google.maps.Polygon({
        paths: instrument.polygons,
        strokeColor: instrument.color,
        strokeOpacity: 0.8,
        visible:false,
        strokeWeight: 1,
        fillColor: instrument.color,
        fillOpacity: 0.35,
        instrumentId: instrument.id,
        map: map,
```

```
        });

        polygons.push(currentPolygon);

        const currentMarker = new google.maps.Marker({
            map,
            position: {
                lat: instrument.pointer_latitude,
                lng: instrument.pointer_longitude
            },
            instrumentId: instrument.id,
            icon:{
                path: 'M12 2c3.196 0 6 2.618 6 5.602 0 2.238-1.058 3.488-
2.659 5.381-1.078 1.274-2.303 2.722-3.341 4.697-1.038-1.976-2.263-3.423-
3.341-4.697-1.601-1.893-2.659-3.143-2.659-5.381 0-2.984 2.804-5.602 6-
5.602zm0-2c-4.198 0-8 3.403-8 7.602 0 6.243 6.377 6.903 8 16.398 1.623-
9.495 8-10.155 8-16.398 0-4.199-3.801-7.602-8-7.602zm0 11c-1.657 0-3-1.343-
3-3s1.343-3 3-3 3 1.343 3 3-1.343 3-3 3z',


                fillColor: instrument.color,
                fillOpacity: 1.0,
                strokeColor: '#000000',
                strokeWeight: 0,
                scale: 2,
                anchor: new google.maps.Point(12, 24),
            },
            optimized: false,
        });

        markers.push(currentMarker);
        google.maps.event.addListener(currentMarker, "click",
        (event) => {
            hideAllPolygons();
            infoWindow.close();
            showInstrumentPolygonById(currentMarker.instrumentId);
            showInstrumentMarkerInfoWindowById(currentMarker.instrumentId);
            closeSearch();
            setSearchPanelVisibility(false);
            closeInstrumentPanel();
        });

        google.maps.event.addListener(currentMarker, "mouseover",
        (event) => {
            hideAllPolygons();
            infoWindow.close();
            showInstrumentPolygonById(currentMarker.instrumentId);
            showInstrumentMarkerInfoWindowById(currentMarker.instrumentId);
            closeSearch();
            setSearchPanelVisibility(false);
            closeInstrumentPanel();
        });
    });

    Array.from(instrumentPanelItems).forEach((element) => {
        element.addEventListener('click', (event) => {
            let instrumentId
                = event.target.getAttribute("data-instrument-id");
            hideAllPolygons();
            infoWindow.close();
            showInstrumentPolygonById();
            showInstrumentMarkerInfoWindowById(instrumentId);
            closeSearch();
```

```
            setSearchPanelVisibility(false);
        })
    })
}


instrumentPanelDropdown.addEventListener('click', (event) => {
    if (isInstrumentPanelOpen){
        instrumentPanelContent.classList.add('d-none');
        isInstrumentPanelOpen = false;
    }else{
        instrumentPanelContent.classList.remove('d-none');
        isInstrumentPanelOpen = true;
        closeSearch();
    }
});
```

HTML

HTML a "HyperText Markup Language" rövidítése, ami olyan szabványosított nyelv, amelyet a világháló (World Wide Web) oldalainak készítésekor használnak.

A HTML alapvetően szöveges kód, aminek a segítségével az oldalak tartalmát strukturáltan lehet megjeleníteni, és a felhasználók számára navigációs elemeket is lehet biztosítani. A HTML különböző elemei (például a fejléc, szöveg, kép, videó stb.) tagekkel vannak megjelölve, amelyeknek különböző tulajdonságokat lehet adni a megjelenésük és viselkedésük finomhangolásához.

A HTML az internetes fejlesztési folyamat alapvető eleme, és együttműködik más technológiákkal, például a CSS-szel (Cascading Style Sheets) és a JavaScripttel, ezek segítségével a megjelenést és az interakciót tovább finomíthatjuk.

CSS

A CSS (Cascading Style Sheets) olyan nyelv, amely lehetőséget ad a weboldalak, webalkalmazások kinézetének, megjelenítésének, elrendezéseinek meghatározására. Elkülöníti a dokumentum tartalmát és formázását egymástól, így lehetővé teszi a weboldal tartalmának és stílusának külön-külön való változtatását, ez javítja az újrafelhasználhatóságot, és könnyíti a karbantartást. A CSS használatával a weboldalak stílusa egyszerűen módosítható, mert a szabályai egyszerű és könnyen értelmezhető szintaxissal rendelkeznek. Lehetőség van az elrendezés, szövegformázás, betűtípusok, méret- és színbeállítások, háttérképek, színek, borítóképek, egyéb stíluselemek meghatározására.

HTML kódban a “style” elemmel illeszthetők be, vagy külső CSS file-okban tárolhatók és hivatkozhatunk rájuk a HTML kódban. Használata elengedhetetlen a modern webes tervezésben, mert lehetőség van vele a weboldalak és alkalmazások testreszabására, és javítja a felhasználói élményt is.

CSS média queryk

Ezekkel érhető el, hogy az alkalmazásunk, weboldalunk dinamikusan reagáljon a felhasználó által használt eszköz képernyőméretére, az azon megjelenő tartalomra. Lehetővé teszik, hogy különböző módon jelenjenek meg az alkalmazás, weboldal tartalmai más-más eszközökön, így segítve a felhasználót a weboldalak és alkalmazások könnyű és hatékony használatához.
Ezek általában a CSS stíluslapokban találhatók, és a képernyőméret vagy eszköztípus alapján változtatják a megjelenített stílusokat. Például, ha weboldalt készítünk, amit asztali és mobil eszközön is szeretnénk használhatóvá tenni, akkor a média queryk segítenek ennek megvalósításában azzal, hogy dinamikusan változtatják az elrendezést és a megjelenő tartalmakat.
A media queryk Gyakorlati bemutatása céljából az előbbiekből kettőt emelek ki:

```
@media only screen and (max-width:720px) {
    .modal-mid-info-name {
        font-weight: 600;
        overflow: hidden;
        margin: 0;
    }
    .modal-mid-info-type-name {
        font-size: 1.5rem;
        font-weight: 400;
        overflow: hidden;
    }
    .modal-mid-info-origin-name {
        font-size: 1.2rem;
        font-weight: 300;
        text-transform: capitalize;
        text-align: start;
        margin: 0;
    }
    .modal-mid-info-origin {
        font-size: 1.2rem;
        font-weight: 300;
        text-align: start;
        margin-bottom: 12px;
    }
}
```

Ez egy CSS media query, ami adott eszközön vagy képernyőméreten belüli megjelenítés-tulajdonságokat tartalmaz. A fenti kód a “screen” típusú kijelzőkre vonatkozik, ezek maximális szélessége 720 pixel, így erre a méretre optimalizálja a megjelenést.

A media query az “only” kulcsszóval kezdődik, ami kizárólagosan erre a méretre vonatkozik, tehát a nagyobb képernyőméretekre nem lesz hatással.
Ezután az adott stílusosztályokat határozzuk meg, amiket a képernyő méretétől függően kell alkalmazni. A fenti példában a modal ablakban szereplő szövegek (nevek, típus, eredet stb.) stílusa és elrendezése van függővé téve a képernyő méretétől, és annak alapján változnak.

```
@media only screen and (max-width:930px) {
    #search, .drop {
        width: calc(100vw - 20px);
        overflow: scroll;
    }
    #search-floating-panel {
        top: 60px;
    }
    #instrument-floating-panel{
        width: 170px;
    }
}
```

Ez egy másik példa a CSS média query használatára. Itt az általános szabály az, hogy a stílusok csak azokon a képernyőkön jelennek meg, amelyek maximális szélessége 930 pixel, erre optimalizálja őket.
A fenti kódrészletben a “#screen” és “.drop” elemek szélessége az eszköz szélességében változik, és az “overflow:scroll” tulajdonsággal érhető el a görgetősáv használata a tartalom megjelenítéséhez.
A “#instrument-floating-panel” elem szélessége 170 pixelre van állítva, ami változtatja a szélességet a képernyő kisebb méretéhez igazítva.
A “#search-floating-panel” elem pozíciója a “top: 60px” értékre van állítva, ez azt jelenti, hogy az elem a felhasználói felület felső szélének 60 pixelére helyezkedik el.

```
function initMap() {
    infoWindow = new google.maps.InfoWindow();
    map = new google.maps.Map(mapElement, {
        center: { lat: 48.5731, lng: 7.5043 },
        zoom: 5,
        streetViewControl: false,
        fullscreenControl: false,
        minZoom: 2,
        restriction: {
            latLngBounds: {
                north: 85,
                south: -85,
                west: -180,
                east: 180
            }
        },
    });

    mapElement.addEventListener("mouseup", (event) => {
```

```
            hideAllPolygons();
            setSearchPanelVisibility(true);
            closeInstrumentPanel();
    });


    mapElement.addEventListener("touchstart", (event) => {
            hideAllPolygons();
            setSearchPanelVisibility(true);
            closeInstrumentPanel();
    });

    instrumentsJson.forEach((instrument) => {
        let currentPolygon = new google.maps.Polygon({
            paths: instrument.polygons,
            strokeColor: instrument.color,
            strokeOpacity: 0.8,
            visible:false,
            strokeWeight: 1,
            fillColor: instrument.color,
            fillOpacity: 0.35,
            instrumentId: instrument.id,
            map: map,
        });

        polygons.push(currentPolygon);

        const currentMarker = new google.maps.Marker({
            map,
            position: {
                lat: instrument.pointer_latitude,
                lng: instrument.pointer_longitude
            },
            instrumentId: instrument.id,
            icon:{
                path: 'M12 2c3.196 0 6 2.618 6 5.602 0 2.238-1.058 3.488-
2.659 5.381-1.078 1.274-2.303 2.722-3.341 4.697-1.038-1.976-2.263-3.423-
3.341-4.697-1.601-1.893-2.659-3.143-2.659-5.381 0-2.984 2.804-5.602 6-
5.602zm0-2c-4.198 0-8 3.403-8 7.602 0 6.243 6.377 6.903 8 16.398 1.623-
9.495 8-10.155 8-16.398 0-4.199-3.801-7.602-8-7.602zm0 11c-1.657 0-3-1.343-
3-3s1.343-3 3-3 3 1.343 3 3-1.343 3-3 3z',


                fillColor: instrument.color,
                fillOpacity: 1.0,
                strokeColor: '#000000',
                strokeWeight: 0,
                scale: 2,
                anchor: new google.maps.Point(12, 24),
            },
            optimized: false,
        });

        markers.push(currentMarker);
        google.maps.event.addListener(currentMarker, "click",
        (event) => {
            hideAllPolygons();
            infoWindow.close();
            showInstrumentPolygonById(currentMarker.instrumentId);
            showInstrumentMarkerInfoWindowById(currentMarker.instrumentId);
            closeSearch();
            setSearchPanelVisibility(false);
            closeInstrumentPanel();
```

```
        });

        google.maps.event.addListener(currentMarker, "mouseover",
        (event) => {
            hideAllPolygons();
            infoWindow.close();
            showInstrumentPolygonById(currentMarker.instrumentId);
            showInstrumentMarkerInfoWindowById(currentMarker.instrumentId);
            closeSearch();
            setSearchPanelVisibility(false);
            closeInstrumentPanel();
        });
    });

    Array.from(instrumentPanelItems).forEach((element) => {
        element.addEventListener('click', (event) => {
            let instrumentId
                = event.target.getAttribute("data-instrument-id");
            hideAllPolygons();
            infoWindow.close();
            showInstrumentPolygonById();
            showInstrumentMarkerInfoWindowById(instrumentId);
            closeSearch();
            setSearchPanelVisibility(false);
        })
    })
}
```

Ez egy JavaScript függvény, amely inicializál egy Google Térkép példányt a középponttal, nagyítási szinttel és más beállításokkal. A függvény hozzáad eseménykezelőket is a térképhez és a markerekhez[24], illetve poligonokat és markereket hoz létre a JSON objektumban lévő adatok alapján.

A függvény definiál egy térképobjektumot, ami a Google Maps API egy példánya. Ezt középponttal és nagyítási szinttel hozzák létre, illetve más beállításokat is eszközölnek, amilyen például a StreetViewControl, fullscreenControl, minZoom és restriction.

Ezután a függvény eseménykezelőket ad a térképelemhez a mouseup és a touchstart eseményekre, ezek elrejtik a megjelenített poligonokat és a keresőpanelt, illetve bezárják az eszköztárat, ha nyitva van, amikor a felhasználó kattintja vagy érinti a térképet. Ezt követően a függvény végigmegy egy JSON változóban meghatározott eszközök tömbjén, amit a blade továbbít az alábbi Script segítségével, amely a Blade json függvényét használja:

```
<script>
    let instrumentsJson = @json($instruments);
</script>
```

Minden eszközre létrehoznak egy poligont es egy markert a térképen. Ezeknek olyan speciális tulajdonságaik vannak, mint a pozíció, a szín, az átlátszóság. Ezek az eszközobjektumban vannak definiálva.

Minden markerhez eseménykezelőket adnak hozzá a kattintás és a mouseover eseményekre, ezek jelenítik meg a poligonokat és az egyéni információs adatokat, illetve elrejtik a megjelenített poligonokat, keresőpanelt és eszköztárat.
Végül eseménykezelőket adnak az eszközökhöz az eszköztárban a kattintás eseményre, amik megjelenítik a poligonokat, információs ablakokat az adott hangszerhez, illetve elrejtik a magjelenített poligonokat és a keresőpanelt.

```
instrumentPanelDropdown.addEventListener('click', (event) => {
    if (isInstrumentPanelOpen){
        instrumentPanelContent.classList.add('d-none');
        isInstrumentPanelOpen = false;
    }else{
        instrumentPanelContent.classList.remove('d-none');
        isInstrumentPanelOpen = true;
        closeSearch();
    }
});
```

Ez a kód nyitja meg vagy zárja be a hangszerpanel felugró paneljét, attól függően, hogy éppen nyitva vagy zárva van. Az instrumentPanelDropdown változó egy gombra vonatkozik, erre kattintva megjelenik vagy eltűnik a panel. Az addEventListener metódus a gombhoz eseménykezelőket rendel, ami a felhasználó ide kattintásakor fut le. Ha az instrumentPanelOpen változó értéke “true”, tehát a panel nyitva van, akkor a kódban lévő “if” ág fut le, és az instrumentPanelContent elem “d-none” osztályát hozzáadjuk, ez eltávolítja a panel láthatóságát. Az instrumentPanelOpen változó értékét “false”-ra állítjuk, ezzel jelezve, hogy a panel mostantól zárva van.
Ha az instrumentPanelOpen változó értéke “false”, tehát a panel még nincs nyitva, akkor a kódban levő “else” ág fut le, ekkor eltávolítjuk a “d-none” osztályt az instrumentPanelContent elemből, hogy megjelenítsük a panelt, aztán az instrumentPanelOpen változó értéke “true”-ra változik, ezzel jelezve, hogy a panel mostantól nyitott állapotban van. Emellett meghívjuk a closeSearch() függvényt, ami a keresési funkció bezárásáért felelős.

```
function closeInstrumentPanel(){
    instrumentPanelContent.classList.add('d-none');
    isInstrumentPanelOpen = false;
}
```

Ez a függvény arra jó, hogy bezárja a hangszerpanel felugró paneljét, ha az nyitva van. Amikor meghívjuk a függvényt, akkor az instrumentPanelContent elemhez hozzáadja a “d-none” osztályt, ami eltávolítja az elem láthatóságát. Az instrumentPanelOpen változó értékét “false”ra állítja, jelezve, hogy a panel mostantól zárva van. Ez a függvény egyszerűsíti a hangszerpanel

zárását, illetve felhasználóbarátabbá teszi az alkalmazást. Máshol is előfordulhat a meghívása, ha például az alkalmazás más részén van szükség a panel bezárására.

Keresés:

```
searchInput.addEventListener('input', (event) => {
    const userInput = event.target.value.toLowerCase()
    closeInstrumentPanel();
    if(userInput.length === 0) {
        searchInput.style.borderBottomLeftRadius = '5px';
        searchInput.style.borderBottomRightRadius = '5px';
        searchDropArea.classList.add('d-none');
        return searchDropArea.innerHTML = '';
    }

    const filteredInstruments = instrumentsJson.filter(
        instrumentsJson =>
        nstrumentsJson.name.toLowerCase().includes(userInput)) ||
        instrumentsJson.filter(instrumentsJson =>
        instrumentsJson.description.toLowerCase()
        .includes(userInput)).sort().splice(0, 5)

    searchDropArea.innerHTML = '';
    let buildResultList = '';
    filteredInstruments.forEach(instrument => {
        buildResultList += getSearchItemContent(instrument);
    })

    searchDropArea.innerHTML = buildResultList;

    if (filteredInstruments.length > 0){
        searchDropArea.classList.add('d-none');
        infoWindow.close();
        hideAllPolygons();
    }

    filteredInstruments.forEach(instrument => {
        const dropItem =
        document.querySelector('[data-instrument-id="'+instrument.id+'"]')
        if (dropItem){
            dropItem.addEventListener('click', (event) => {
                let instrumentId =
        event.target.closest('[data-instrument-id="'+instrument.id+'"]')
        .getAttribute("data-instrument-id");
                hideAllPolygons();
                infoWindow.close();
                showInstrumentPolygonById(instrumentId);
                showInstrumentMarkerInfoWindowById(instrumentId);
                setSearchPanelVisibility(false);
                closeSearch();
            });
        }
    })
    if (filteredInstruments.length > 0){
        searchInput.style.borderBottomLeftRadius = '0';
        searchInput.style.borderBottomRightRadius = '0';
        searchDropArea.classList.remove('d-none');
    }else{
        searchInput.style.borderBottomLeftRadius = '5px';
        searchInput.style.borderBottomRightRadius = '5px';
        searchDropArea.classList.add('d-none');
    }
});
```

A fenti kódrészlet egy eseménykezelőt tartalmaz, amely az input mező tartalmának változásakor fut le. Az eseménykezelő beolvassa az input mezőben megadott felhasználói inputot, ezután meghatározza, hogy mely hangszerhez tartozó bejegyzések felelnek meg a megadott szűrésnek.

A függvény első lépésben bezárja a hangszerpanel felugró paneljét a closeInstrumentPanel() függvénnyel, ezután ellenőrzi, hogy az input mező üres-e. Ha igen, törli az összes bejegyzést a searchDropArea elemről, eltávolítja az alsó keret lekerekítését, és elrejti a searchDropArea elemet. Ha nem üres, akkor az eseménykezelő meghatározza a felhasználó által megadott szűrésnek megfelelő hangszereket az instrumentsJson tömbből. A szűrés a hangszer nevére és a leírására vonatkozhat, a találatokat azonnal megjeleníti a searchDropArea elemen.

A filteredInstruments tömbben található valamennyi hangszerhez létrehoz egy listaelemet, és ezeket hozzáadja a searchDropArea elemhez. Ezután az eseménykezelő meghatározza, hogy a találatokat megjelenítő searchDropArea elem látható-e. Ha igen, eltávolítja az alsó keret lekerekítését az input mezőről, hogy egybeolvadjon a keresési eredményekkel. Ha a találatok üresek, a searchDropArea elem van elrejtve, és a beállítások visszaállnak az eredeti értékeikre. Végül az eseménykezelő minden talált elemhez hozzáad egy eseményfigyelőt, ami akkor fut le, ha a felhasználó rákattint az adott hangszer nevére. Ha a felhasználó rákattint egy elemre, a függvény bezárja a hangszerlista paneljét, és elrejti a keresési eredményeket, ezután meghívja az infoWindow[25] és a polygons tömbök által reprezentált markerhez és poligonhoz tartozó különféle függvényeket a megfelelő hangszerhez.

```
function closeSearch(){
    searchInput.style.borderBottomLeftRadius = '5px';
    searchInput.style.borderBottomRightRadius = '5px';
    searchDropArea.innerHTML = '';
    searchDropArea.classList.add('d-none');
}
```

Ez a függvény visszaállítja a keresőmező szegélyének sugarára az eredeti értéket, és törli a keresési eredménylistát, tehát a searchDropArea elem innerHTML értékét üres stringre állítja. Emellett elrejti a keresési eredménylistát, hozzáadva a „d-none” osztályt az elemhez.

```
function setSearchPanelVisibility(show){
    if (window.innerWidth < 930) {
        if (show){
            searchAreaPanel.classList.remove('d-none');
        }else{
            searchAreaPanel.classList.add('d-none');
        }
    }
}
```

Ez a függvény a *show* paraméter értékének megfelelően állítja be a keresőpanel láthatóságát. Ha az ablak szélessége kisebb, mint 930 pixel, akkor vagy eltávolítja, vagy hozzáadja a „d-none” osztályt a searchAreaPanel elemhez.

Általános funkciók

```
function getInstrumentById (id){
    return instrumentsJson.find(x => x.id === id);
}
```

Ez a függvény adja vissza azokat a hangszerobjektumokat az instrumentsJson tömbből, amelyeknek az id tulajdonsága megegyezik az id paraméter értékével.

Markerek kezelése:

```
function showInstrumentMarkerInfoWindowById (id){
    let marker = markers.find(x => x.instrumentId == id);
    let instrument = getInstrumentById(marker.instrumentId);
    infoWindow.setContent(getInfoWindowContent(instrument));
    infoWindow.open(marker.getMap(), marker);
}
```

Ez a függvény megjelenít egy infowindow-t a térképen, ami megjeleníti az adott hangszert a markerrel. A függvény először keresi a megfelelő markert az id paraméter értékével, ezután az instrumentId tulajdonságának felhasználásával meghatározza az instrument objektumot a getInstrumentById függvénnyel. Ezek után létrehozza az infowindow tartalmát a getInfoWindowContent függvény segítségével, beállítja az infowindow tartalmát, és megnyitja az infowindow-t a megfelelő markerrel a marker.getMap() metódusával.

Poligonok

```
function showInstrumentPolygonById (id){
  let polygon = polygons.find(x => x.instrumentId == id);
  polygon.setVisible(true);
```

A showInstrumentPolygonById függvény a polygons tömbben keresi azt a poligont, ami az adott id-hoz tartozik, ezután láthatóvá teszi azt a térképen a setVisible metódus segítségével. Például ha így hívjuk meg:
showInstrumentPolygonById('123'), akkor az 123 instrumentId tulajdonságú poligon láthatóvá válik a térképen

```
function hideAllPolygons (){
  polygons.forEach((polygon) => {
    polygon.setVisible(false);
```

```
    })
}
```

A hideAllPolygons függvény a polygons tömbökön végighaladva minden poligonra meghívja a setVisible metódust, illetve beállítja a láthatóságát false-ra, így az összes poligont eltünteti.

Diszkusszió, továbblépési lehetőségek

Oktatási célzatú továbblépés

Az adatbázis és a téma kiterjeszthető több ütőhangszerre, illetve más hangszercsaládokra is, például billentyűs, fúvós, vonós, egyéb húros, pengetős hangszerekre. Ezek tematikus használata zenei, zenetörténeti illetve szolfézsórákon megkönnyíti a tanulók számára a megértést.

Többféle tematikus csoportosítás is lehetséges, például hangszerek „megszületése” alapján, vagy olyan megközelítésben, hogy az egyes hangszereket milyen művészeti korszakban használták, illetve mikor terjedtek el nagyobb területen (reneszánsz, barokk, klasszika, romantika stb.).

Tartalmi továbblépés

Jelen szakdolgozat inkább a megvalósíthatóság problémáját hivatott kifejteni, megválaszolni. A tartalmi fejlődés területe összefügg az oktatási célzatú továbblépéssel, hangszerenként több adat, részletesebb, tartalmasabb leírás segítségével még hasznosabb oktatási segédanyag állítható elő.

Technikai, technológiai továbblépési lehetőségek

Technikai, technológia továbblépési lehetőségként többek között elképzelhető például telefonos applikáció fejlesztése, vagy akár az adatkészlet letölthetővé tétele offline felhasználás biztosítása céljából.

Összefoglalás

Az általam készített oldal létrehozása és ismertetése volt a feladatom. Fő célom egy olyan rendszer létrehozása volt, amely online térképen jeleníti meg a bemutatni kívánt ütőhangszereket, illetve információkat tartalmaz azok elterjedéséről, használatáról.
A rendszer fejlesztése során rengeteget fejlődtem, és jelentős szakmai előrelépést értem el. Ez volt az első olyan projektem, ahol Laravel keretrendszert használtam, ennek a megismerése nagy mértékű segítséget nyújtott a projekt írása közben, így már saját tapasztalatból mondhatom, hogy rendkívülien felhasználóbarát. A legnagyobb fejtörést az okozta, hogy az infobox megjelenítésekor a poligonok egy része ki volt takarva, erre kellett megoldást találni, de ez is sikerült az infobox kétlépésessé tételével.
Az alapkoncepciót egyes esetekben videós példákkal egészítettem ki, sajnos nem minden hangszer esetében lehet olyat találni, ami megfelelően bemutatná a hangszer felhasználását, de fejlesztési szempontból izgalmas volt kihasználni a YouTube-videók beágyazásának lehetőségét.
Továbbfejlesztés céljából rengeteg ötlet merült még fel, ezek közül az egyik legjelentősebb, hogy egy-egy hangszerhez több képet is lehessen feltölteni, például ha a tradicionális és a modernebb, elterjedtebb hangszer szignifikánsabban különbözik.
Meglátásom szerint tudásomhoz mérten jól sikerült megoldanom az általam célul kitűzött feladatot, kicsit talán nagy fába vágtam a fejszémet, de alapvetően elégedett vagyok a projekt jelenlegi állapotával is. Szándékozom a jövőben geoinformatikával foglalkozni, ehhez szerintem sikerült elég jó alaptudást szereznem ezzel a projekttel, egy ismeretterjesztésre mindenképpen alkalmas oldal létrehozásával.
Mivel a tanulmányaim nem fedik le teljesen ezt a területet, sok mindennek utánanéztem, hogy hogyan kellene megcsinálni, amit hasznosnak találok az esedékes jövőbeli projektjeimhez. Alapvetően izgalmas és érdekes volt elkészíteni a térképet, és több szempontból is hasznosnak és értékesnek ítélem meg az elkészült projektet, amelyben további tartalombővítésre és fejlesztésre is vannak még lehetőségek.

Köszönetnyilvánítás

Szeretném megköszönni konzulensemnek, Dr. Gede Mátyásnak az útmutatást és a téma támogatását, külsős konzulensemnek és ütőhangszeres tanáromnak, Holló Miklósnak a kutatás segítését és az információk ellenőrzését, barátomnak, Dudás Miklósnak a szakmai segítséget, illetve családtagjaimnak az értelmezhetőség ellenőrzését és a szöveg korrektúrázását.

Irodalomjegyzék

Poligonok [1], [2]

https://developers.google.com/maps/documentation/javascript/reference/polygon#Polygon

https://developers.google.com/maps/documentation/javascript/examples/polygon-arrays

Múzeum Digitár: [3]

https://hu.museum-digital.org/oak?gesusa=1398#map=1.94/6266631.60/564699.51/0

OpenStreetMap [4]

https://www.openstreetmap.org/about

Historical-basemaps adatkészletet: [5]

https://github.com/aourednik/historical-basemaps

Keyhole Markup Language: [6]

https://developers.google.com/kml

Extensible Markup Language [7]

https://www.w3schools.com/xml/xml_whatis.asp

PHP [8]

https://www.php.net/

Laravel [9], [10]

https://laravel.com/

https://laravel.com/docs/9.x

Laravel ORM, Eloquent [11], [20]

https://laravel.com/docs/9.x/eloquent

Natív SQL [12]

https://www.techtarget.com/searchdatamanagement/definition/SQL

MVC [13]

https://developer.mozilla.org/en-US/docs/Glossary/MVC

Bootstrap [14]

https://getbootstrap.com/docs/5.2

Tárhelyszolgáltatók [15]

https://cweb.hu/ugyfeladmin/index.php?rp=/login

MySQL [16]

https://www.mysql.com/

Composer [17]

https://getcomposer.org/

Adatbázis [18]

https://dbdiagram.io/home

Infowindow, Youtube [19]

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_youtubeiframe

Insomnia [21]

https://insomnia.rest/

Színek [22]

https://htmlcolorcodes.com/color-picker/

Event [23]

https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseover_event

https://www.w3schools.com/jsref/dom_obj_event.asp

Marker [24]

https://medium.com/free-code-camp/how-to-change-javascript-google-map-marker-color-8a72131d1207

Infowindow [25]

https://developers.google.com/maps/documentation/javascript/infowindows

Források:

Ulrich Michels – Springer Hungarica Atlasz – Zene

Tamburin (csörgődob):

https://www.wpr.org/shows/tambourine-and-music-making-around-globe