Универзитет „ Св. Кирил и Методиј “- Скопје
Факултет за информатички науки и компјутерско инженерство

Семинарска работа по предметот Имплементација на системи со отворен код

Визуелизација на 3Д податоци во web browser

Ментори:

д-р Иван Чорбев

м-р Бобан Јоксимоски

Студент:

Вероника Ѓорева 151151

Февруари 2019

Содржина

Вовед

Во денешно време со напредокот на технологијата, добиваме се поголем број на алатки кои би можеле да ги искористеме за прикажување и споделување на разновидна содржина. Целта на овие алатки е да се создаде квалитетна содржина која би го задржало вниманието на корисниците, но во исто време би ги натерало повторно да се вратат на конкретната веб страна. Една таква алатка е една од библиотеките на JavaScript наречена Three.js. Таа претставува библиотека која се користи за создавање и прикажување на анимирани 3Д објекти во веб пребарувачите со користење само на JavaScript јазикот, без додавање на дополнителни додатоци за пребарувачите. Ова е овозможено благодарение на доаѓањето на WebGL (Web Graphics Library), кое што е JavaScript API, иницијално достапно за јавноста од 3-ти Март, 2017 година. Некои од функционалностите што ги нуди Three.js се создавање на сцени во рамките на веб пребарувачот, додавање на ефекти, дефинирање на камери со кои корисникот би можел да ги следи објектите во рамките на сцената во која се наоѓаат, додавање на анимација, светла, создавање на објекти или прикачување на веќе постоечки објекти кои што корисникот ги има на својот компјутер.

Токму оваа алатка, во комбинација со Laravel, одлучив да ја искористам при изработката на оваа веб апликација. Станува збор за веб апликација која овозможува продажба на музички инструменти, каде инструментите претставуваат 3Д објекти со цел подобро да се долови изгледот на инструментите кои корисникот би сакал да ги купи. Самата апликација е достапна за сите, такашто нема креирање на кориснички профили, туку за секој инструмент што се чува во базата, се чува и енкодирана трансакциска сметка на сопсвеникот што го објавил огласот.

Инсталација на Laravel и креирање на иницијалните табели

За да можеме да започнеме со развојот на веб апликацијата, најпрво е потребно да се инсталира Laravel framework-от. Тоа се прави со користење на терминалот на PhpStorm, каде што ја впишуваме следната команда:

composer create-project –prefer-dist laravel/laravel quais-una-fantasia

Со помош на оваа команда се создава нашиот иницијален проект со име quasi-una-fantasia и во исто време се инсталира Laravel со сите нејзини зависности. Откако ќе заврши инсталацијата, го отвараме проектот и во .env датотеката ги впишуваме името на нашата локална база на податоци, која што ќе ја користеме во апликацијата, корисничкото име и лозинката со кои ќе се овозможи пристап до базата.

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=musicalinstruments
DB_USERNAME=151151
DB_PASSWORD=Lozinka151151
```

По конектирањето со локалната база на податоци, може да се започне со креирањето на моделите. Оваа апликација има вкупно три модели и четири табели, поточно табелите create_users_table и create_passwords_resets_table се генерираат автоматски од страна на Laravel откако прв пат ќе ја извршеме командата

php artisan migrate

која всушност служи за прикачување на локално создадените табели во локалната база на податоци. Останатите две табели се:

2019_01_24_002446_instruments_family.php

2019_01_24_003731_instruments_info.php

кои се генерираат со создавањето на соодветните модели InstrumentsInfo.php и InstrumentsFamily.php. Модели се создаваат со помош на командата

php artisan make:model nameOfModel -m

Иницијализација на табелите

Во табелата *2019_01_24_002446_instruments_family.php* чуваме податоци за четирите фамилии инструменти: Ударни, Жичани, Дрвени дувачки и Бакарни дувачки.

```
public function up()
{
    Schema::create('instruments_family', function (Blueprint $table) {
        $table->increments('id');
        $table->string('family')->unique();
       # $table->timestamps();
    });

    DB::table('instruments_family')->insert([
        'family' => "Brass"
    ]);

    DB::table('instruments_family')->insert([
        'family' => "Persussion"
    ]);

    DB::table('instruments_family')->insert([
        'family' => "Strings"
    ]);

    DB::table('instruments_family')->insert([
        'family' => "Woodwind"
    ]);
}
```

Додека во табелата *2019_01_24_003731_instruments_info.php* се чуваат информации за секој поединечен инструмент.

```
Schema::create('instruments_info', function (Blueprint $table) {
    $table->string('id');
    $table->primary('id');
    $table->string('name')->unique();
    $table->text('details');
    $table->string('credit_card_number');
    $table->decimal('price', 10, 2);
    $table->string('object');
    $table->string('preview');
    $table->unsignedInteger('in_store');
    $table->unsignedInteger('instrument_family');
    $table->foreign('instrument_family')->references('id')->on('instruments_family')
        ->onDelete('cascade')
        ->onUpdate('cascade');
    $table->timestamps();
});
```

Примарниот клуч во instruments_info рачно го дефинираме при внесувањето на податоците во табелата. Поточно за генерирање на примарниот клуч се користи Webpatser\Uuid\Uuid пакетот, кој може да се додаде на проектот со помош на

composer require "webpatser/laravel-uuid:^3.0"

При секое внесување на нов инструмент во базата се користи Uuid::generate()->string функцијата, со што се добива уникатен идентификатор, со кој се заштитува редниот број на инструментот при GET повиците при користењето на апликацијата. Двете табели се поврзуваат преку примарниот клуч на instruments_family табелата, кој што е надворешен клуч во instruments_info табелата. На овој начин можат меѓусебно да комуницираат и да се прстапуваат одредени податоци од едната табела кога се познати некои од податоците од другата табела.

Иницијалните податоци во instruments_info табелата се дефинирани во up() функцијата, каде за сите вредности кои ги дефиниравме при создавањето на табелата, ги внесуваме соодветните податоци. За да можеме да пристапеме до објектите и сликите кои се сместени во public >> models папката, во базата чуваме информација за нивните релативни патеки. Бројот на кредитната картичка е енкодиран во базата за да има дополнителна заштита корисникот што го објавил огласот за дадениот инструмент.

```
DB::table('instruments_info')->insert([
    'id' => \Webpatser\Uuid\Uuid::generate()->string,
    'object' => 'models/acoustic_guitar/scene.gltf',
    'preview' =>'models/acoustic_guitar/img.png',
    'name' => "Yamaha FG800",
    'credit_card_number' => base64_encode(378282246310005),
    'details' => "A standard design and a traditional look come together in one of the
best rated acoustic guitars which is available at a very affordable price. The Yamaha
FG800 gives you comfort in tune with your needs. This is another addition to Yamaha's
esteemed FG series. The focus was always to produce a high-quality musical instrument
with explicit playability and outstanding tuning without a punching a hole in your
wallet. The FG series has always been considered a classic choice amongst millions of
guitarists for the past 50 years. The FG800 is a continuation of the series and offers
even more refined playability along with features that are a class apart. It has
always been a symbol of assurance to professionals and has never failed to strike the
perfect tone of expectation with contemporary musicians.",
    'price' => 56.08,
    'in_store' => 8,
    'instrument_family' => 3,
    'created_at' => now(),
    'updated_at' => now()
]);
```

Овие податоци се внесуваат во нашата локална база на податоци со командата

php artisan migrate:fresh

која ги отстранува досега внесените податоци и повторно ги прикачува нашите табели и ги пополнува со соодветно дефинираните информации.

Иницијализација на моделите

Како што кажавме погоре, моделите се генерираат во исто време со нашите табели, па така имаме три модели. Поточно едната се нарекува User која се генерира автоматски од страна на Laravel заедно со create_users_table, а останатите два модела се оние кои што ние ги дефинираме, односно InstrumentsFamily и InstrumentsInfo. Во моделите ја дефинираме променливата $fillable со која наведуваме кои полиња можеме да ги пополнуваме со наши вредности, кои ни слижат најчесто при POST повиците.

Во InstrumentsFamily можеме да додаваме вредности само во family полето, кое што е име на фамилијата од инструменти. Втората променлива $table ја означува табелата на која се однесува конкретниот модел. Во овој случај станува збор за instruments_family табелата.

```
class InstrumentsFamily extends Model
{
    protected $fillable = array('family');
    protected $table = 'instruments_family';
}
```

Во InstrumentsInfo ги внесуваме сите полиња што ги дефинираме во табелата, затоа што понатаму кога ќе додаваме нов инструмент, ги внесуваме сите полиња. Променливата $table за овој модел е поврзана со instruments_info табелата. Во овој модел дефинираме врска со останатите два модела преку посебни функции, односно family() функцијата ги поврзува instruments_info со instruments_family, додека user() функцијата ја поврзува instruments_info со create_users_table.

```
class InstrumentsInfo extends Model
{
    protected $fillable = ['id', 'name', 'details', 'credit_card_number','object',
'price', 'in_store', 'preview', 'created_at', 'updated_at', 'instrument_family'];
    protected $table = 'instruments_info';

    public function family() {
        return $this->belongsTo(InstrumentsFamily::class);
    }

    public function user(){
        return $this->belongsTo(User::class);
    }
}
```

Дефинирање на контролери

За потребите на нашата веб апликација, создадени се три контролери: InstrumentController, UploadController и CartController, со помош на кои се извршуваат главните функционалности, при комуникација со погледите и табелите.

a) InstrumentsController ги содржи следниве функции:

- **home()**

  Содржи една променлива $post која од InstrumentsInfo табелата ќе ги земе најновите три инструменти и ги враќа погледот кон почетната страна и променливата $post, за да може да се пристапат овие податоци во погледот.

```
public function home() {
    $post = InstrumentsInfo::orderBy('created_at','desc')->take(3)->get();

    return view('home', ['post' => $post]);
}
```

- **instruments($id)**

  Содржи две променливи $posts и $data. Во $posts се чуваат сите инструменти од InstrumentsInfo кои припаѓаат на фамилијата $id, подредени во опаѓачки редослед, според времето кога е објавен нивниот оглас. На еден поглед може да има истовремено 9 инструменти. Променливата $data е низа, која чува информации за индексот на фамилијата и ја чува променилвата $posts. Оваа функција го враќа Каталог погледот, заедно со $data променливата.

```
public function instruments($id) {

    $posts = InstrumentsInfo::where('instrument_family' ,$id)-
>orderBy('created_at','desc')->paginate(9);

    $data = array('family' => $id, 'instruments' => $posts );

    return view('instruments.catalogue')->with($data);
}
```

- **instrument_info($id, $index)**

  Содржи три променливи $instrument, $family и $data, каде $instrument го чува инструментот со индекс $index, $family го чува името на фамилијата инструменти со индекс $id, а $data е помошна променлива што ги содржи претходните две променливи. Функцијата враќа поглед кон бараниот инструмент на кој можат да се видат подетални информации за истиот, заедно со $data променливата.

```php
public function instrument_info($id, $index) {

    $instrument = InstrumentsInfo::find($index);

    $family = InstrumentsFamily::find($id);

    $data = array('instrument' => $instrument, 'family_name' => $family);

    return view('instruments.instrument_info')->with($data);
}
```

- **instrument_info_home($id, $index)**
  Оваа функција ја има истата структура како и instrument_info() функцијата, со таа разлика што не води кон поглед сличен поглед како и претходниот, но кога корисникот ќе го одбере копчето за назад, тој ќе се врати на почетната страна.

- **instrument_info_preview($id, $index)**
  Оваа функција ја има истата структура како и претходните две функции, со тоа што не води кон поглед што е исти како погледите на претходните два, но не содржи копче за додавање на инструментот во кошничката и кога корисникот ќе го одбере копчето за назад, ќе се врати во кошничката.

b) UploadController се користи при додавање на нов инструмент во базата на податоци, односно кога се објавува нов оглас. Во него го правиме повикот за прикажување на погледот кон погледот за создавање на нов оглас, како и валидацијата на информациите и прикачување на објектите и сликите од страна на корисникот. Тој ги содржи следниве функции:

- **create()**
  Во оваа функција имаме само една променлива $family која ги чува сите податоци од табелата instrument_family и го враќа погледот кон формата за објавување на нов оглас, заедно со $family променливата

```php
public function create(){

    $family = InstrumentsFamily::all();

    return view('upload', ['family' => $family]);
}
```

- **upload(Request $request)**
  Овде се справуваме со податоците што ќе ги внесе корисникот при создавањето на нов оглас. Најпрво имаме валидација на податоците што ќе се внесат, пример дали името е уникатно, дали цената и достапната количина на инструменти се броеви, дали сите полиња се пополнети. Потоа со имплементација на пакетот за валидација на кредитни картички

Inacho\CreditCard, ги проверуваме бројот на кредитната картичка, нејзиниот рок на истекување и CVC бројот дали се валидни. Следна проверка е дали имаме прикачено податоци од нашиот компјутер, односно дали корисникот ги прикачил потребниот 3Д модел од инструментот, слика од инструментот и текстура/и за. Потребно е да се прикачат минимум четири вакви податоци (најмалку една слика што претставува текстура). Доколку некои од овие барања не е исполнето се појавува порака што му укажува на корисникот што не е во ред со податоците што ги внесел. Кога сите барања ќе бидат исполнети, најпрво се справуваме со податоците што се прикачени, односно се создава нов директориум во public >> models според името на инструментот. Потоа сите податоци се преместуваат од storage директориумот во ново создадениот директориум. Податоците што имаат екстензии .gltf и .bin се преименуваат во scene.gltf и scene.bin за GLTFLoader-от на Three.js да може полесно да ги вчитува. Сите текстури (сликите што не се викаат img.png) се преместуваат во папка наречена textures што се наоѓа во нашиот директориум. Последен чекор во оваа функција е да се зачиваат овие податоци во локална база на податоци. Откако ќе се внесат во базата, се пренасочува кон погледот на новиот оглас.

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

    $request->validate([
        'name' => 'required | unique:instruments_info',
        'details' => 'required',
        'price' => 'required | numeric',
        'in_store' => 'required | numeric',
        'files' => 'required'
    ]);

    $datetime = explode('-', $request->get('expiration_date'));
    $card = $request->card_number;

    if(CreditCard::validCreditCard($card)['valid'] == 0)
        return back()->with('warning', 'Invalid credit card number!');
    else if(!CreditCard::validCvc($request->cvc,
CreditCard::validCreditCard($card)['type']))
        return back()->with('warning', 'Invalid CVC!');
    else if(CreditCard::validDate($datetime[0], $datetime[1]) == false)
        return back()->with('warning', 'Expired card!');

    $nameOfFolder = $request->name;
   // $flag = true;


    if($request->hasFile('files')) {

        if(sizeof($request->file('files')) < 4){
            return back()->with('success', 'Minimum 4 files are required!');
        }
        else{
            foreach ($request->file('files') as $file) {
```

```
                $filename = $file->getClientOriginalName();

                $name_temp = explode('.', $filename);

                $extension = $name_temp[1];

                if ($extension == 'gltf') {
                    $file->move(public_path('/models/' . $nameOfFolder),
'scene.gltf');
                } else if ($extension == 'bin') {
                    $file->move(public_path('/models/' . $nameOfFolder),
'scene.bin');
                } else if ($filename == 'img.png') {

                    $file->move(public_path('/models/' . $nameOfFolder),
$filename);
                } else {
                    $file->move(public_path('/models/' . $nameOfFolder .
'/textures'), $filename);
                }

            }
        }
    }

    $instrument = new InstrumentsInfo;
    $instrument->id = \Webpatser\Uuid\Uuid::generate()->string;
    $instrument->name = $request->name;
    $instrument->credit_card_number = base64_encode($request->card_number);
    $instrument->details = $request->details;
    $instrument->price = $request->price;
    $instrument->in_store = $request->in_store;
    $instrument->instrument_family = $request->instrument_family;
    $instrument->created_at = now();
    $instrument->updated_at = now();
    $instrument->object = 'models/'.$nameOfFolder.'/scene.gltf';
    $instrument->preview ='models/'.$nameOfFolder.'/img.png';
    $instrument->save();
    $instrumentIndes = InstrumentsInfo::where('name', $request->name)-
>first();

    return redirect('/catalogue/'.$request-
>instrument_family.'/instruments/'.$instrumentIndes->id);
}
```

c) CartController се справува со сите функционалности поврзани со кошничката. Во овој контролер е имплементиран Gloudemans\Shoppingcart\Facades\Cart пакетот со кој лесно можеме да се справуваме со содржината во кошничката. Ги содржи следниве функции:

- **index()**
  Враќа поглед кон кошничката и две променливи $data (инструментите во кошничката) и $total (вкупната цена).

```
public function index()
{
    $data = Cart::content();
    $total = Cart::subtotal();
    return view('shopping_cart', ['data' => $data, 'total' => $total]);
}
```

- **increase($id)**

  Го наоѓа инструментот што се наоѓа на $id позицијата во кошничката и ја зголемува количината. Доколку ја достигне границата на достапни инструменти во тој оглас се враќа порака дека не може да се зголемува повеќе.

```
public function increase($id)
{
    $instrument = InstrumentsInfo::find($id);
    $rows  = Cart::content();
    $rowID = $rows->where('id', $id)->first()->rowId;
    $cartInstrument = Cart::get($rowID);

    if($cartInstrument->qty < $instrument['in_store']){
        $qty = $cartInstrument->qty + 1;
        Cart::update($rowID, $qty);
    }
    else{
        return back()->with('warning', 'You have reached the limit of
available instruments!');
    }

    return back();
}
```

- **decrease($id)**

  Го наоѓа инструментот што се наоѓа на $id позицијата во кошничката и ја намалува количината. Доколку количината е 1 тогаш ќе се врати порака дека не е можно понатамошно намалување на количината.

```
public function decrease($id)
{
    $rows  = Cart::content();
    $rowID = $rows->where('id', $id)->first()->rowId;
    $cartInstrument = Cart::get($rowID);

    if($cartInstrument->qty > 1){
        $qty = $cartInstrument->qty - 1;
        Cart::update($rowID, $qty);
    }
    else{
        return back()->with('warning', 'Cannot have a quantity lower than
1!');
    }

    return back();
}
```

- **store(Request $request)**
  Оваа функција се повикува во погледот за прикажување на податоците во врска со даден инструмент и со неа се додава инструментот во кошничката. Доколку е внесен инструментот во кошничката, при обид повторно да се внесе во кошничката ќе се појави соодветна порака.

```
public function store(Request $request)
{
    $duplicates = Cart::search(function ($cartItem, $rowId) use ($request) {
        return $cartItem->id === $request->id;
    });
    if (!$duplicates->isEmpty()) {
        return back()->with('warning', 'Item is already in your cart!');
    }
    Cart::add($request->id, $request->name, 1, $request->price)-
>associate('App\InstrumentsInfo');
    return back()->with('success', 'Item was added to your cart!');
}
```

- **remove($id)**
  Го отстранува инструментот од кошничката што се наоѓа на $id-тата позиција во кошничката

```
public function remove($id)
{
    $rows  = Cart::content();
    $rowID = $rows->where('id', $id)->first()->rowId;

    Cart::remove($rowID);
    return redirect('shopping_cart')->with('success', 'Item has been
removed!');
}
```

- **clear()**
  Ги отстранува сите инструменти од кошничката

```
public function clear(){
    Cart::destroy();
    return redirect('shopping_cart')->with('success', 'Cart has been
cleared!');
}
```

- **checkout()**
  Го враќа погледот за плаќање

```
public function checkout(){
    $data = Cart::content();
    $total = Cart::subtotal();
    return view('checkout', ['data' => $data, 'total' => $total]);
}
```

- **validateCheckout(Request $request)**
  Се прават приближно истите проверки како во UploadController-от при внесувањето на нови инструменти, со таа разлика што откако ќе завршат сите проверки, соодветно се додаваат промените за in_store полето. Доколку количината на достапни инструменти на даден оглас е 0, тој се отстранува од базата на податоци. На крајот се испразнува кошничката.

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

    $request->validate([
        'full_name' => 'required',
        'email' => 'required',
        'city' => 'required',
        'province' => 'required',
        'postal_code' => 'required | numeric',
        'phone' => 'required | numeric',
        'name_on_card' => 'required'
    ]);

    if(strlen($request->phone) != 8)
        return redirect('checkout')->with('danger', 'Invalid phone
number!');

    $datetime = explode('-', $request->get('expiration_date'));
    $card = $request->card_number;

    if(CreditCard::validCreditCard($card)['valid'] == 0)
        return back()->with('danger', 'Invalid credit card number!');
    else if(!CreditCard::validCvc($request->cvc,
CreditCard::validCreditCard($card)['type']))
        return back()->with('danger', 'Invalid CVC!');
    else if(CreditCard::validDate($datetime[0], $datetime[1]) == false)
        return back()->with('danger', 'Expired card!');

    foreach(Cart::content() as $item){

        $flag = false;
        $instrument = InstrumentsInfo::find($item->id);
        $instrument->in_store = $instrument->in_store - $item->qty;
        if($instrument->in_store == 0)
            $flag = true;
        $instrument->save();
        if($flag)
            InstrumentsInfo::find($item->id)->delete();
    }

    Cart::destroy();
    return redirect('shopping_cart')->with('success', 'Payment
successful!');
}
```

Дефинирање на патеки и погледи

Во routes >> web.php ги дефинираме сите патеки што ни се потребни за навигација помеѓу погледите и за испраќање на податоци од формуларите што ги имаме во погледот за создавање на нов оглас и погледот за плаќање на инструментите во кошничката.

```
Route::get('/', function () {
    return redirect('/home');
});

Route::get('/home', 'InstrumentController@home')->name('home');

Route::get('/catalogue', function () {
    return redirect('/catalogue/1');
});

Route::get('/catalogue/{id}', 'InstrumentController@instruments');

Route::get('/catalogue/{id}/instruments/{index}',
'InstrumentController@instrument_info');

Route::get('/catalogue/{id}/latest/{index}',
'InstrumentController@instrument_info_home');

Route::get('/catalogue/{id}/preview/{index}',
'InstrumentController@instrument_info_preview');

Route::get('/upload', 'UploadController@create');

Route::post('upload', 'UploadController@upload')->name('upload');

Route::get('/shopping_cart', 'CartController@index');

Route::post('/shopping_cart', 'CartController@store');

Route::get('/shopping_cart/increase/{id}', 'CartController@increase');

Route::get('/shopping_cart/decrease/{id}', 'CartController@decrease');

Route::get('/shopping_cart/remove/{id}', 'CartController@remove');

Route::get('/shopping_cart/clear', 'CartController@clear');

Route::get('/shopping_cart/checkout', 'CartController@checkout');

Route::post('/shopping_cart/checkout', 'CartController@validateCheckout');
```

Погледите се наоѓаат во resources >> views. За потребите на нашата веб апликација имаме девет погледи, меѓу кои customPagination.blade.php е изгледот на копчињата за страничење во каталогот. Сите погледи се статични страни, чии изглед се добива со помош на bootstrap и css. Најгоре имаме навигација со која можеме да пристапеме до четири главни страни: почетна, каталог, додавање на нов оглас и кошничката.

a) **home.blade.php**
   Го содржи насловот и описот на страната и 3-те најнови инструменти.

b) **catalogue.blade.php**
   Иницијално ги прикажува инструментите од Brass фамилијата на инструменти. Доколку корисникот сака да ги виде инструментите од останатите фамилии може да ја одбере посакуваната фамилија од навигацијата. Во овој поглед имаме страничење доколку во одредена фамилија имаме повеќе од девет инструменти.

c) **instruments_info.blade.php, instruments_info_home.blade.php и instruments_info_preview.blade.php**
   Овие три погледи се исти по структура и во него имаме div дел каде се прикажува 3Д моделот на инструментот со помош на Three.js. Инструментот содржи текстури и можеме да го зголемуваме или намалуваме и да го ротираме за да можеме да добиеме подобра претстава за изгледот на инструментот што би биле заинтересирани да го купеме. Исто така имаме опис на инструментот, како и податоци за неговата цена, фамилијата на инструменти во која припаѓа, колку инструменти од овој тип се достапни.

d) **upload.blade.php**
   Овој поглед содржи формулар во кој се пополнуваат сите полиња според натписот што го имаат дадено.

e) **shopping_cart.blade.php**
   Овој поглед содржи табеларен приказ на инструментите што се наоѓаат во кошничката. Доколку е празна кошничката, се појавува само div елемент со натпис дека кошничката е празна.

f) **checkout.blade.php**
   Овој поглед од левата страна содржи формулар со полиња што треба да се пополнат, а од десната страна имаме табеларен приказ на инструментите што ќе ги купи корисникот

Имплементација на Three.js

Three.js го искористив за прикажување на инструментите како 3Д модели, со цел корисниците да имаат подобра претстава за изгледот на инструментите, како и нивниот квалитет, за разлика од стандардниот начин со гледање на повеќе слики од различни агли.

Во трите погледи за прегледот на деталите на одреден инструмент го имаме

* *

во кој ја сместуваме нашата сцена. Во script тагот дефинираме најпрво main(id) функција. Овде најпрво се поврзуваме со нашиот div таг преку id-то што го внесуваме во main функцијата. Потоа создаваме canvas, ја дефинираме неговата големина и ја додаваме на нашиот div. Во main функцијата дефинираме низа од елементи, како: рендер, камерата и нејзината позиција, контролите за управување на камерата во рамикте на canvas-от, креирање на сцена и соодветни светла со кои би можеле подобро да го гледаме објектот. Во рамките на main функцијата имаме и вгнездена функција frameArea со помош на која ја позиционираме камерата секогаш да се центрира на објектите, без разлика на нивната оригинална големина. Со THREE.GTLFLoader() функцијата, ја вчитуваме релативната патека на дадениот објект и се вчитува објектот заедно со неговите текстури.

```
var obj = String('{{asset($instrument['object'])}}');
const gltfLoader = new THREE.GLTFLoader();
gltfLoader.load(obj, (gltf) => {
    const root = gltf.scene;
    scene.add(root);

    // compute the box that contains all the stuff
    // from root and below
    const box = new THREE.Box3().setFromObject(root);

    const boxSize = box.getSize(new THREE.Vector3()).length();
    const boxCenter = box.getCenter(new THREE.Vector3());

    // set the camera to frame the box
    frameArea(boxSize * 0.5, boxSize, boxCenter, camera);

    // update the Trackball controls to handle the new size
    controls.maxDistance = boxSize * 10;
    controls.target.copy(boxCenter);
    controls.update();
});
```

Откако ќе ги дефинираме сите потребни функции за прикажување на 3Д моделот, во script делот ја повикуваме main функцијата и и го внесуваме името на id-то на нашиот div каде што треба да се смести објектот. Крајниот резултат е инструмент кој можеме да го доближеме до нас или да го оддалечеме, и кој можеме да го видеме од сите страни.

Заклучок

Како што секојдневно напреднува технологијата, не е ни чудно што голем број на веб програмери започнуваат да користат технологии како Three.js во своите апликации. Причината не е само креативноста и интересниот изглед, туку и интерактивноста што им се нуди на корисниците што добиваат можност да видат таква апликација. Quasi Una Fantasia е еден обичен пример за како понатаму би можел да биде светот за продажба преку интернет, но не само во таа област туку и во секоја од останатите области. Големите компании како Facebook веќе се на пат кон развој на плтаформа за разговор помеѓу пријателите во виртуелна околина, што уште повеќе ни укажува на фактот дека виртуелниот свет се повеќе присутен. Како ќе изгледа тоа во иднина? Само времето ќе ни покаже.

Користена литература

- Three.js
  https://en.wikipedia.org/wiki/Three.js
- WebGL
  https://en.wikipedia.org/wiki/WebGL
- Credit Card validator
  https://packalyst.com/index.php/packages/package/inacho/php-credit-card-validator
- Three.js load gltf
  https://threejsfundamentals.org/threejs/lessons/threejs-load-gltf.html