**После обновления migration таблиц базы выполнить следующее:**

1. Сделать миграцию и посеять seed

php artisan migrate:fresh --seed

1. Для правильной работы пакета passport Auth2 , выполнить команду

// установить параметры passport в базе

php artisan passport:install

php7.3 artisan passport:install

1. Проверить работу superviser – постановка задач и работа с очередями

Локально из под приложения запуск планировщика

php artisan schedule:run

php7.3 artisan schedule:run

на сервере проверить открытость порта и если нет то открыть

открыть порт на сервере - sudo iptables -I INPUT -p tcp -m tcp --dport 6001 -j ACCEPT

открытые порты - sudo lsof -i -P -n | grep LISTEN

>>>

выполнение задач

php artisan queue:work --queue=database

php7.3 artisan queue:work --queue=database

>>>

рестарт задач после изменений в php

php artisan queue:restart

>>>

**После перезагрузки сервака – открыть порт для Socket**

sudo iptables -I INPUT -p tcp -m tcp --dport 6001 -j ACCEPT

инфо для открытия на постоянно: <https://forum.ubuntu.ru/index.php?topic=249178.0>

>>>

**Рабочая версия npm i socket.io@1.7.4**

**\package.json**

"dependencies": {  
"ioredis": "^4.22.0",  
"socket.io": "^1.7.4",

}

**Реализация отправки данных в socket при добавлении нового игрока в сессию. (Аттракциону)**

Сокращенно – без логики

app\Http\Controllers\SessionController.php

public function update(UpdateSessionRequest $request) {  
  
 // добавить игрока в сессию  
 $session = $this->addGamerSession($index);  
  
}

\app\Http\Traits\SessionTraite.php

protected function addGamerSession($index){  
  
 $this->addGamerUpdateSession($session, $index);  
  
}

private function addGamerUpdateSession($session, $index){  
  
 event(new AddGamer([$index['attraction\_id'], $gamer]));  
  
}

\app\Events\AddGamer.php

public $attraction\_id;  
public $gamer;  
  
public function \_\_construct($data) {  
 $this->attraction\_id = $data[0];  
 $this->gamer = $data[1];  
}  
  
public function broadcastOn() {  
 return ['attraction'];  
}  
  
// Имя события  
public function broadcastAs() {  
 return 'add-gamer\_'.$this->attraction\_id;  
}  
  
// приходит в событии  
public function broadcastWith() {  
 return [  
 'gamer' => $this->gamer,  
 ];  
}

\ws.server\server.js

// сервер websocket  
let app = require("express")();  
let http = require("http").Server(app).listen(6001, function () {  
 ***console***.log("listening in port 6001");  
});  
let io = require("socket.io")(http);  
  
// Redis слушает события Laravel приложения  
var Redis = require('ioredis');  
//клиент приложения  
var redis = new Redis();  
  
// подписаться на события  
// \* все события  
// error - ошибка в базе данных  
// count - кол-во подписанных событий этого клиента  
redis.psubscribe('\*', function(error, count) { });  
  
// всплытие событий от Laravel  
// pattern - шаблон события  
// channel - канал события Laravel  
// message - сообщение события Laravel  
redis.on('pmessage', function(pattern, channel, message) {  
  
 parsedMessage = ***JSON***.parse(message);  
 //console.log(parsedMessage);  
  
 if (parsedMessage.event.indexOf('add-gamer') !== -1){  
 if (typeof parsedMessage.data.gamer.user\_id !== 'undefined'){  
 io.emit( parsedMessage.event,  
 { data: "{user\_id:" + parsedMessage.data.gamer.user\_id + ", nickname:" + parsedMessage.data.gamer.user\_one.nickname + "}" }  
 );  
 }  
 // для получения Test данных - не обьект user  
 else{  
 io.emit( parsedMessage.event, message);  
 }  
 }  
});

**Локально сервер запускается в консоле**

node ws.server/server.js

**на сервере в supervisor – файл конфига команды**

**команда для запуска node сервера -** node ws.server/server.js

создать файл - etc/supervisor/conf.d/node-server.conf

[program:node-server]

command=sudo node ws.server/server.js

directory = /home/admin/web/ch-throw.ru/public\_html

user=root

autostart=true

autorestart=true

startretries=3

stdout\_logfile=/home/admin/web/ch-throw.ru/public\_html/storage/logs/node\_server.log

stdout\_logfile\_maxbytes=1MB

stdout\_logfile\_backups=10

stopsignal=TERM

environment=NODE\_ENV=production

numprocs=1

>>>

**Реализация отправки данных в socket при изменении статуса (Закончилась) лицензии администратора аттракциона. (в 24 часа ночи аттрациону)**

1. Команда постановки задач запускается локально php artisan schedule:run

Или на сервере supervisor

создать файл - etc/supervisor/conf.d/shedul-laravel.conf

**Запуск с перерывом в 60 сек**

[program:shedul-laravel]

command=/bin/sh -c "while [ true ]; do (php /home/admin/web/ch-throw.ru/public\_html/artisan schedule:run --verbose --no-interaction &); sleep 60; done"

user=root

autostart=true

autorestart=true

numprocs=1

redirect\_stderr=true

stdout\_logfile=/home/admin/web/ch-throw.ru/public\_html/storage/logs/supervisor\_queue-work.log

В самом Kernel настройка времени постановки задачи может варьироваться.

\app\Console\Kernel.php

protected function schedule(Schedule $schedule) {  
// каждый день в 1 мин первого ночи по серверному времени  
$schedule->call(new LicenseConsole)->dailyAt('00:01');

}

1. Файл постановки задач

\app\Console\LicenseConsole.php

public function \_\_invoke() {  
 // отправить в очередь задачу с паузой в 5 сек (драйвер в настройках - database)  
 SendLicenseJobs::*dispatch*()->onQueue('database')->delay(5);

}

1. Файл выполнения задачи

app\Jobs\SendLicenseJobs.php

Выполнение задач в очереди

Не отлавливает изменения в задаче

php artisan queue:work --queue=database

Отлавливает изменения в задаче

php artisan queue:listen --queue=database

class SendLicenseJobs implements ShouldQueue {  
 use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;  
  
 // задержка повторной попытки неудачного задания  
 public $retryAfter = 30;  
 // количество ошибочных выполнений задачи  
 public $tries = 3;  
  
  
public function \_\_construct($int) {  
 // назначить этой задаче имя очереди  
 $this->onQueue('database');  
 }  
  
public function handle() {  
  
 $coll = (new LicenseRepository)->get();  
 $day\_now = Carbon::*today*()->toDateString();  
  
 foreach ($coll as $key => $license){  
  
 // лицензия не пуста  
 if(!is\_null($license->month)){  
 // проверка в наступившие 00:01 часа ночи  
 // прошел месяц и 1 день  
 $lic\_month = Carbon::*parse*($license->month)->addMonth()->addDay()->toDateString();  
 // лицензия протухла  
 if ($lic\_month == $day\_now) {  
 event(new SendLicenseEvent([$license->attraction\_id, 'stop']));  
 }  
 }  
 }  
  
 logs()->info("lic = ", (Array)$coll);  
 logs()->info("Заливка Локализации в базу завершена");  
 }  
  
  
 // метод вызывается при ошибке выполнения задачи  
 // $exception - передает сообщение об ошибке  
 public function failed(\Throwable $e) {  
 logs()->info("Ошибка задачи : AddLocalizationData ");  
 logs()->info($e->getMessage());  
 }  
}

1. Файл Event - \app\Events\SendLicenseEvent.php

class SendLicenseEvent implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $attraction\_id;  
 public $status;  
  
public function \_\_construct($data) {  
 $this->attraction\_id = $data[0];  
 $this->status = $data[1];  
 }  
  
public function broadcastOn() {  
 return ['license'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'license-status\_'.$this->attraction\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'attraction\_id' => $this->attraction\_id,  
 'status' => $this->status,  
 ];  
 }  
}

1. Файл сервера node - \ws.server\server.js

Команда запуска сервера

node ws.server/server.js

// сервер websocket  
let app = require("express")();  
let http = require("http").Server(app).listen(6001, function () {  
 ***console***.log("listening in port 6001");  
});  
let io = require("socket.io")(http);  
  
// Redis слушает события Laravel приложения  
var Redis = require('ioredis');  
//клиент приложения  
var redis = new Redis();  
  
// подписаться на события  
// \* все события  
// error - ошибка в базе данных  
// count - кол-во подписанных событий этого клиента  
redis.psubscribe('\*', function(error, count) { });  
  
// всплытие событий от Laravel  
// pattern - шаблон события  
// channel - канал события Laravel  
// message - сообщение события Laravel  
redis.on('pmessage', function(pattern, channel, message) {  
  
 parsedMessage = ***JSON***.parse(message);  
 //console.log(parsedMessage);  
  
 if (parsedMessage.event.indexOf('license-status') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{user\_id:" + parsedMessage.data.user\_id + ", license-status:" + parsedMessage.data.message + "}" }  
 );  
 }  
});

1. Файл клиента - \resources\views\welcome.blade.php

{{-- библиотека socket.io --}}  
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 var socket = io("{{ env('APP\_URL') }}:6001");

// 1 подписка на event и принятие данных

socket.on('license-status\_10000', function(data){  
 ***console***.log(data);  
 });  
</script>

**Реализация отправки данных в socket игроку при активации, закрытии и удалении сессии – (срабытывает в 3 событиях). (Игроку)**

Файл \app\Events\SessionStatusEvent.php

class SessionStatusEvent implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $session\_id;  
 public $status;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->session\_id = $data[0];  
 $this->status = $data[1];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['session-status'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'session-status'.$this->session\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'session\_id' => $this->session\_id,  
 'status' => $this->status,  
 ];  
 }  
}

файл \app\Http\Traits\SessionTraite.php

// ========================================  
 // активировать сессию  
 protected function ActivateTraiteSession($index){  
 // выслать socket в клиент о запуске сессии  
 event(new SessionStatusEvent([$index['session\_id'], 'start']));  
 return true;  
 }

// ========================================  
// удаляет указанную сессию  
protected function deleteSession($session\_id){  
 // выслать socket в клиент о запуске сессии  
 event(new SessionStatusEvent([$session\_id, 'close']));  
return $session;  
}

// ========================================  
// закрывает указанную сессию  
protected function closeSession($index){  
 // выслать socket в клиент о запуске сессии  
 event(new SessionStatusEvent([$index['session\_id'], 'stop']));  
return $session;  
}

Файл \ws.server\server.js

node ws.server/server.js

// сервер websocket  
let app = require("session-status")();  
let http = require("http").Server(app).listen(6001, function () {  
 ***console***.log("listening in port 6001");  
});  
let io = require("socket.io")(http);  
  
// Redis слушает события Laravel приложения  
var Redis = require('ioredis');  
//клиент приложения  
var redis = new Redis();  
  
// подписаться на события  
// \* все события  
// error - ошибка в базе данных  
// count - кол-во подписанных событий этого клиента  
redis.psubscribe('\*', function(error, count) { });  
  
// всплытие событий от Laravel  
// pattern - шаблон события  
// channel - канал события Laravel  
// message - сообщение события Laravel  
redis.on('pmessage', function(pattern, channel, message) {  
  
 parsedMessage = ***JSON***.parse(message);  
  
 if (parsedMessage.event.indexOf('session-status') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{session\_id:" + parsedMessage.data.session\_id + ", status:" + parsedMessage.data.status + "}" }  
 );  
 }  
});

Файл \resources\views\welcome.blade.php

{{-- библиотека socket.io --}}  
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");

socket.on('session-status\_1', function(data){  
 ***console***.log(data);  
 });  
</script>

**Реализация отправки данных в socket при переходе хода игрока в сессии. (игроку)**

Файл \app\Events\SessionPlayerTransitEvent.php

class SessionPlayerTransitEvent implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $session\_id;  
 public $json;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->session\_id = $data[0];  
 $this->json = $data[1];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['session-transit'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'session-transit\_'.$this->session\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'session\_id' => $this->session\_id,  
 'data' => $this->json,  
 ];  
 }  
}

файл \app\Http\Traits\SessionTraite.php

// ========================================  
// переход хода игрока  
protected function PlayerTransitSession($index){  
  
 $auth = Auth::*user*();  
 // владелец аттракциона или root (мои доступы, admin\_id)  
 $this->service->checkOwnerAdmin($auth, $index['admin\_id']);  
  
 // отправить json от аттракциона юзеру  
 event(new SessionPlayerTransitEvent([$index['session\_id'], $index['data']]));  
return true;  
}

Файл \ws.server\server.js

node ws.server/server.js

// сервер websocket  
let app = require("session-status")();  
let http = require("http").Server(app).listen(6001, function () {  
 ***console***.log("listening in port 6001");  
});  
let io = require("socket.io")(http);  
  
// Redis слушает события Laravel приложения  
var Redis = require('ioredis');  
//клиент приложения  
var redis = new Redis();  
  
// подписаться на события  
// \* все события  
// error - ошибка в базе данных  
// count - кол-во подписанных событий этого клиента  
redis.psubscribe('\*', function(error, count) { });  
  
// всплытие событий от Laravel  
// pattern - шаблон события  
// channel - канал события Laravel  
// message - сообщение события Laravel  
redis.on('pmessage', function(pattern, channel, message) {  
  
 parsedMessage = ***JSON***.parse(message);  
  
 if (parsedMessage.event.indexOf('session-transit') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{session\_id:" + parsedMessage.data.session\_id + ", data:" + parsedMessage.data.data + "}" }  
 );  
 }

});

Файл \resources\views\welcome.blade.php

{{-- библиотека socket.io --}}  
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");

socket.on('session-transit\_1', function(data){  
 ***console***.log(data);  
});

</script>

**Реализация выход игрока из игры и отправка в socket на аттракцион кого нужно удалить из сессии. (аттракцион)**

Файл \routes\api.php

Route::*prefix*('session')->group( function () {  
 // завершение игры игроком  
 Route::*post*('/player\_exit', 'SessionController@playerExit');  
});

Файл \app\Http\Controllers\SessionController.php

// завершение игры игроком  
public function playerExit(PlayerExitRequest $request){  
 $index = $request->validated();  
 $bool = $this->PlayerExitSession($index);  
return response()->json( $bool , 201);  
}

Файл \app\Http\Traits\SessionTraite.php

// завершение игры игроком  
protected function PlayerExitSession($index){  
 $auth = Auth::*user*();  
 // Я игрок и переданый id мой  
 $this->service->checkGamerId($auth, $index['gamer\_id']);  
 // отправить socket в аттракцион  
 event(new ExitGamer([$index['session\_id'], $index['gamer\_id']]));  
return true;  
}

Файл \app\Events\ExitGamer.php

class ExitGamer implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $session\_id;  
 public $gamer\_id;  
  
public function \_\_construct($data) {  
 $this->session\_id = $data[0];  
 $this->gamer\_id = $data[1];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*/* public function broadcastOn() {  
 return ['gamer-status'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'gamer-status\_'.$this->session\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'gamer\_id' => $this->gamer\_id,  
 'session\_id' => $this->session\_id,  
 ];  
 }  
}

Файл \ws.server\server.js

node ws.server/server.js

if (parsedMessage.event.indexOf('gamer-status') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{session\_id:" + parsedMessage.data.session\_id + ", gamer\_id:" + parsedMessage.data.gamer\_id + "}" }  
 );  
}

Файл \resources\views\welcome.blade.php

<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");  
  
 socket.on('gamer-status\_1', function(data){  
 ***console***.log(data);  
 });  
</script>

**Реализация отправки данных в socket при запросе на добавление игрока к сессии. (аттракцион)**

Файл \routes\api.php

Route::*prefix*('session')->group( function () {  
// запрос игроком на добавление к сессии  
Route::*post*('player\_start', 'SessionController@playerStart');

});

Файл \app\Http\Controllers\SessionController.php

public function playerStart(PlayerStartRequest $request){  
 $index = $request->validated();  
 $bool = $this->PlayerStartSession($index);  
return response()->json( $bool , 201);  
}

Файл \app\Http\Traits\SessionTraite.php

// завершение игры игроком  
protected function PlayerExitSession($index){  
 $auth = Auth::*user*();  
 // Я игрок и переданый id мой  
 $this->service->checkGamerId($auth, $index['gamer\_id']);  
 // отправить socket в аттракцион  
 event(new ExitGamer([$index['session\_id'], $index['gamer\_id']]));  
return true;  
}

Файл \app\Events\StartGamer.php

class StartGamer implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $session\_id;  
 public $gamer\_id;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->session\_id = $data[0];  
 $this->gamer\_id = $data[1];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['session'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'query\_add\_gamer\_'.$this->session\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'gamer\_id' => $this->gamer\_id,  
 'session\_id' => $this->session\_id,  
 ];  
 }  
}

Файл \ws.server\server.js

node ws.server/server.js

else if (parsedMessage.event.indexOf('query\_add\_gamer') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{session\_id:" + parsedMessage.data.session\_id + ", gamer\_id:" + parsedMessage.data.gamer\_id + "}" }  
 );  
}

Файл \resources\views\welcome.blade.php

<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");  
  
socket.on('query\_add\_gamer\_1', function(data){  
 ***console***.log(data);  
});

</script>

**Реализация отправки данных в socket при добавлении игрока в сессии. (игроку)**

Файл \app\Events\ResponseAddGamer.php

class ResponseAddGamer implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $bool;  
 public $gamer\_id;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->gamer\_id = $data[0];  
 $this->bool = $data[1];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['response\_add\_gamer'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'response\_add\_gamer\_'.$this->gamer\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'user\_is\_added' => $this->bool,  
 ];  
 }  
}

файл \app\Http\Traits\SessionTraite.php

// добавление игрока и обновление токена сессии  
protected function addGamerSession($index){  
// отправить socket игроку  
event(new ResponseAddGamer([$index['gamer\_id'], $bool ]));

return $session;  
}

Файл \ws.server\server.js

node ws.server/server.js

// сервер websocket  
let app = require("session-status")();  
let http = require("http").Server(app).listen(6001, function () {  
 ***console***.log("listening in port 6001");  
});  
let io = require("socket.io")(http);  
  
// Redis слушает события Laravel приложения  
var Redis = require('ioredis');  
//клиент приложения  
var redis = new Redis();  
  
// подписаться на события  
// \* все события  
// error - ошибка в базе данных  
// count - кол-во подписанных событий этого клиента  
redis.psubscribe('\*', function(error, count) { });  
  
// всплытие событий от Laravel  
// pattern - шаблон события  
// channel - канал события Laravel  
// message - сообщение события Laravel  
redis.on('pmessage', function(pattern, channel, message) {  
  
 parsedMessage = ***JSON***.parse(message);  
  
else if (parsedMessage.event.indexOf('response\_add\_gamer') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{user\_is\_added:" + parsedMessage.data.user\_is\_added + "}" }  
 );  
}

});

Файл \resources\views\welcome.blade.php

{{-- библиотека socket.io --}}  
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");

socket.on('response\_add\_gamer\_3', function(data){  
 ***console***.log(data);  
});

</script>

**Socket После удаления игрока отправить игроку статус удаления (игроку)**

Файл \app\Events\DeleteGamer.php

class DeleteGamer implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $gamer\_id;  
 public $delete\_status;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->gamer\_id = $data[0];  
 $this->delete\_status = $data[1];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['delete\_status'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'delete\_status\_'.$this->gamer\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'delete\_status' => $this->delete\_status,  
 ];  
 }  
}

файл \app\Http\Traits\SessionTraite.php

// ========================================  
// удалить игрока из сессии  
protected function deleteGamerSession($index){  
 $bool = false;  
 // если игрок подтвердил на аттракционе удаление себя из сессии  
 if($index['confirmation']){  
 $bool = true;  
 }  
 // выслать socket в клиент о статусе его удаления  
 event(new DeleteGamer([$index['gamer\_id'], $bool]));  
return $session;  
}

Файл \ws.server\server.js

node ws.server/server.js

// сервер websocket  
let app = require("session-status")();  
let http = require("http").Server(app).listen(6001, function () {  
 ***console***.log("listening in port 6001");  
});  
let io = require("socket.io")(http);  
  
// Redis слушает события Laravel приложения  
var Redis = require('ioredis');  
//клиент приложения  
var redis = new Redis();  
  
// подписаться на события  
// \* все события  
// error - ошибка в базе данных  
// count - кол-во подписанных событий этого клиента  
redis.psubscribe('\*', function(error, count) { });  
  
// всплытие событий от Laravel  
// pattern - шаблон события  
// channel - канал события Laravel  
// message - сообщение события Laravel  
redis.on('pmessage', function(pattern, channel, message) {  
  
 parsedMessage = ***JSON***.parse(message);  
  
else if (parsedMessage.event.indexOf('delete\_status') !== -1){  
 io.emit( parsedMessage.event,  
 { data: "{delete\_status:" + parsedMessage.data.delete\_status + "}" }  
 );  
}

});

Файл \resources\views\welcome.blade.php

{{-- библиотека socket.io --}}  
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");

socket.on('delete\_status\_1', function(data){  
 ***console***.log(data);  
});

</script>

**Реализация отправки данных в socket при смене id image игроком. (аттракцион)**

Файл \routes\api.php

// установить картинку для юзера  
Route::*post*('user\_images/set\_id', 'UserImageController@setImage');

Файл \app\Http\Controllers\UserImageController.php

public function setImage(SelectImageRequest $request) {  
 $index = $request->validated();  
 $bool = $this->setImageTrate($index);  
return response()->json( $bool , 201);  
}

Файл \app\Http\Traits\UserImageTraite.php

// ========================================  
// установить картинку для юзера  
protected function setImageTrate($index){  
 // выслать socket аттракциону  
 $this->eventSelectImage($index);  
}

// выслать socket аттракциону  
 private function eventSelectImage($index){  
 event(new ChangeUserImageEvent([  
 $session->session\_id,  
 $index['user\_id'],  
 $select\_image->select\_border,  
 $select\_image->select\_avatar,  
 ]  
 ));  
 }

Файл \app\Events\ChangeUserImageEvent.php

class ChangeUserImageEvent implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $session\_id;  
 public $gamer\_id;  
 public $select\_border;  
 public $select\_avatar;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->session\_id = $data[0];  
 $this->gamer\_id = $data[1];  
 $this->select\_border = $data[2];  
 $this->select\_avatar = $data[3];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['change\_image\_gamer'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'change\_image\_gamer\_'.$this->session\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'gamer\_id' => $this->gamer\_id,  
 'select\_border' => $this->select\_border,  
 'select\_avatar' => $this->select\_avatar,  
 ];  
 }  
}

Файл \ws.server\server.js

node ws.server/server.js

else if (parsedMessage.event.indexOf('change\_image\_gamer') !== -1){  
 io.emit( parsedMessage.event,  
 {  
 data:  
 "{gamer\_id:" + parsedMessage.data.gamer\_id +  
 ", select\_border:" + parsedMessage.data.select\_border +  
 ", select\_avatar:" + parsedMessage.data.select\_avatar +  
 "}"  
 }  
 );  
}

Файл \resources\views\welcome.blade.php

<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");  
  
// socket.on('change\_image\_gamer\_3', function(data){  
// console.log(data);  
// });

</script>

**Реализация отправки данных в socket при смене nickname игроком. (аттракцион)**

Файл \routes\api.php

// обновить данные юзера  
Route::*post*('update\_data', 'AutherController@updateData');

Файл \app\Http\Controllers\Auth\AutherController.php

public function updateData(UpdateDataRequest $request) {  
 $response = $this->changeDataTrait($index);  
}

Файл \app\Http\Traits\AuthTraite.php

// =========================================  
// обновить данные юзера  
public function changeDataTrait($index){  
 // находится ли user в сессии  
 if($session = (new SessionUserRepository())->whereArrFirst([ 'user\_id' => $user->id ])){  
 // отправка в аттракцион новый nickname  
 event(new ChangeUserNicknameEvent([  
 $session->session\_id,  
 $user->id,  
 $index['nickname'],  
 ]  
 ));  
 }  
}

Файл \app\Events\ChangeUserNicknameEvent.php

class ChangeUserNicknameEvent implements ShouldBroadcastNow{  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
  
 public $session\_id;  
 public $user\_id;  
 public $nickname;  
  
 */\*\*  
 \* Create a new event instance.  
 \*  
 \** ***@return*** *void  
 \*/* public function \_\_construct($data) {  
 $this->session\_id = $data[0];  
 $this->user\_id = $data[1];  
 $this->nickname = $data[2];  
 }  
  
 */\*\*  
 \* Имя канала по которому транслируется событие  
 \*  
 \** ***@return*** *\Illuminate\Broadcasting\Channel|array  
 \*/* public function broadcastOn() {  
 return ['change\_user\_nickname'];  
 }  
  
 // Имя события  
 public function broadcastAs() {  
 return 'change\_user\_nickname\_'.$this->session\_id;  
 }  
  
 // приходит в событии  
 public function broadcastWith() {  
 return [  
 'user\_id' => $this->user\_id,  
 'nickname' => $this->nickname,  
 ];  
 }  
}

Файл \ws.server\server.js

node ws.server/server.js

else if (parsedMessage.event.indexOf('change\_user\_nickname') !== -1){  
 io.emit( parsedMessage.event, {  
 data:  
 "{nickname:" + parsedMessage.data.nickname +  
 ", user\_id:" + parsedMessage.data.user\_id + "}"  
 }  
 );  
}

Файл \resources\views\welcome.blade.php

<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>  
<script>  
 // инициализация соединения  
 // node ws.server/server.js  
 var socket = io("{{ env('APP\_URL') }}:6001");  
  
socket.on('change\_user\_nickname\_8', function(data){  
 ***console***.log(data);  
});

</script>

**Реализация достижений у Локации**

**Запрос взятия данных достижений**

POST /api/achievements\_location/select?location\_id=2

Выбирает достижения указанной локации. Выборка отдается игроку и если обновление таблицы были в прошлом часе, то происходит постановка отложенной задачи (пересчет не более чем через 10 сек) пересчет данных с обновлением времени и данных достижений локации.

**Данные для определения достижений:**

Таблица [history\_sessions](http://ch-throw.ru/phpmyadmin/sql.php?db=admin_axes&table=history_sessions&token=0c7ab2ddb237ffa22fcf09a0144f27be) – данные сюда пападают при завершении одной игры. Берутса данные игроков, времени игры, и привязка аттракциона к нужной локации

Таблица [location\_mode\_statistics](http://ch-throw.ru/phpmyadmin/sql.php?db=admin_axes&table=location_mode_statistics&token=0c7ab2ddb237ffa22fcf09a0144f27be) - данные сюда пападают при завершении одной игры. Берутса данные режима игры, локации и кол-во сыгранных в этом режиме игр.

**Как достижения отправляются в responce :**

Если достижения еще не разу не создавалиcm –



Если достижения создавались -



long\_liver - как и остальные 9 полей означают название достижения по примеру таблицы расписанных достижений

'long\_liver'=> [  
 'title\_ru' => 'Долгожитель',  
 'title\_en' => 'Long-liver',  
 'description\_ru' => 'Локация активна :type лет',  
 'description\_en' => 'Location is active for :type years',  
 'type' => [1, 3, 5],  
],  
'axes\_all\_directions'=> [  
 'title\_ru' => 'Топоры во все стороны',  
 'title\_en' => 'Axes in all directions',  
 'description\_ru' => 'На локации сыграно :type игр',  
 'description\_en' => ':type games played at the location',  
 'type' => [1000, 2000, 5000],  
],  
'access\_granted'=> [  
 'title\_ru' => 'Доступ разрешен',  
 'title\_en' => 'Access granted',  
 'description\_ru' => 'На локации зарегистрировались :type пользователей',  
 'description\_en' => ':type players have registered at the location',  
 'type' => [100, 500, 1000],  
],  
'in\_love\_bullseye'=> [  
 'title\_ru' => 'Любовь к Bullseye',  
 'title\_en' => 'In love with Bullseye',  
 'description\_ru' => 'На локации сыграно :type игр Bullseye',  
 'description\_en' => ':type Bullseye games played at the location',  
 'type' => [500, 1000, 2000],  
],  
'in\_love\_sniper'=> [  
 'title\_ru' => 'Любовь к Sniper',  
 'title\_en' => 'In love with Sniper',  
 'description\_ru' => 'На локации сыграно :type игр Sniper',  
 'description\_en' => ':type Sniper games played at the location',  
 'type' => [500, 1000, 2000],  
],  
'in\_love\_throw\_royal'=> [  
 'title\_ru' => 'Любовь к Throw Royal',  
 'title\_en' => 'In love with Throw Royal',  
 'description\_ru' => 'На локации сыграно :type игр Throw Royal',  
 'description\_en' => ':type Throw Royal games played at the location',  
 'type' => [500, 1000, 2000],  
],  
'in\_love\_monopoly'=> [  
 'title\_ru' => 'Любовь к Monopoly',  
 'title\_en' => 'In love with Monopoly',  
 'description\_ru' => 'На локации сыграно :type игр Monopoly',  
 'description\_en' => ':type Monopoly games played at the location',  
 'type' => [500, 1000, 2000],  
],  
'hype'=> [  
 'title\_ru' => 'Ажиотаж',  
 'title\_en' => 'Hype',  
 'description\_ru' => 'Локацию за один день посетило :type зарегистрированных гостей',  
 'description\_en' => 'Location in one day was visited by :type registered guests',  
 'type' => [50, 100, 200],  
],  
'packed\_to\_eyeballs'=> [  
 'title\_ru' => 'Не протолкнуться',  
 'title\_en' => 'Packed to the eyeballs',  
 'description\_ru' => 'Локацию за месяц посетило :type игроков',  
 'description\_en' => 'Location was visited by :type players per month',  
 'type' => [1000, 2000, 5000],  
],  
'who\_you'=> [  
 'title\_ru' => 'Кто вы?',  
 'title\_en' => 'Who are you?',  
 'description\_ru' => 'Локацию за один день посетило :type не зарегистрированных гостей',  
 'description\_en' => 'Location was visited by :type unregistered guests in one day',  
 'type' => [100, 200, 500],  
],

Поле type в достижении означает уровень достижения, всего их 3. В response приходят уже готовые description исходя из type.

**Как протестировать работу обновления достижений.**

1. Понизить type на сервере в файле \config\game\location\_achievements.php
2. Изменить параметры в таблице [location\_mode\_statistics](http://ch-throw.ru/phpmyadmin/sql.php?db=admin_axes&table=location_mode_statistics&token=0c7ab2ddb237ffa22fcf09a0144f27be)



Location\_id – локация с которой вы работаете

Mode\_id – режим игры в котором происходит начисление отыгранных игр

Count\_game – кол-во сыгранных игр в этом режиме

1. В этом файле на сервере - \app\Services\LocationAchievementServices.php

Скопировать и поставить эту строку рядом с return

UpdateLocationAchievementJobs::*dispatch*($index['location\_id'])->onQueue('database')->delay(5);

1. Запустить Post url - /api/achievements\_location/select?location\_id=2

Мы помним что задача выполняется не сразу - подождать 5 сек , после чего запустить url еще раз

ВЕРНУТЬ ВСЕ НА СВОИ МЕСТА !