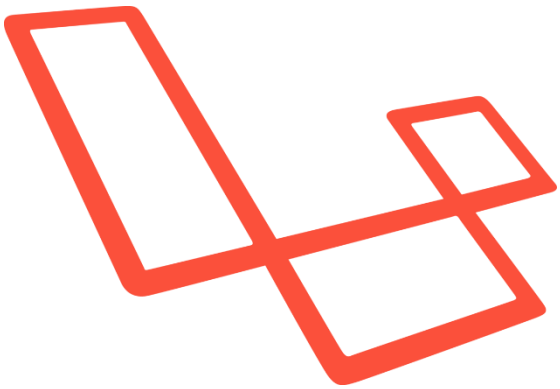
Универзитет „Св. Кирил и Методиј“ – Скопје

Факултет за информатички науки и компјутерско инженерство (ФИНКИ)



Разработка на WebSocket и имплементација во Laravel 5.4

Изработил:

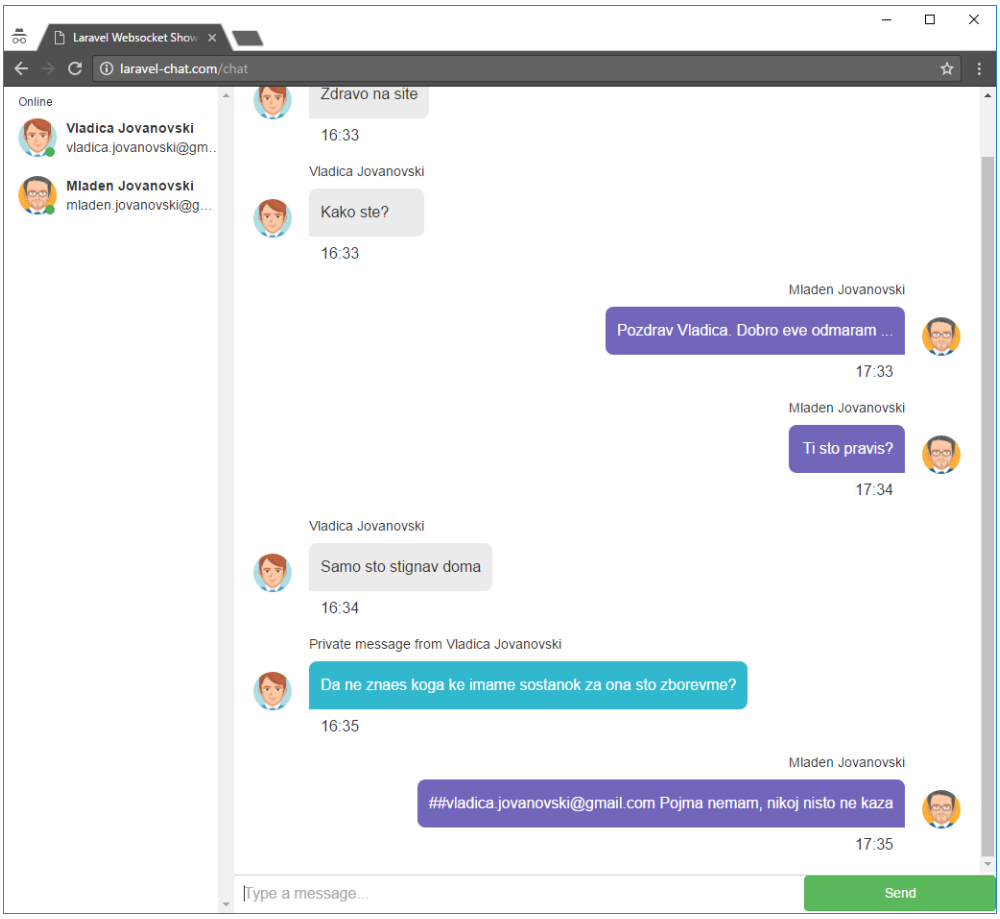
Владица Јовановски 121202

Апстракт

Потребата за динамичко ажурирање на веб апликациите во реално време, особено на нивниот кориснички интерфејс, не е нова. За надминување на овој проблем, но и на голем број други, создаден е протоколот WebSocket кој се базира и е едно ниво погоре над стандардниот TCP протокол. За ситуацијата да стане поинтересна, Apache серверот не нуди нативна поддршка за овој протокол без дополнително инсталирање на модули и екстензии. Посредно, веб апликациите развиени во PHP, Perl, Python и слично кои се извршуваат на Apache серверот мора да бараат други алетернативи за интеграција со WebSocket протоколот. Laravel рамката за развој, во своите последни верзии (во овој момент 5.4), ја воведува Event Broadcasting парадигмата која го користи WebSocket протоколот индиректно. Парадигмата се базира на користење на proxy (посредник) за остварување на WebSocket протоколот со клиентот, како што се Pusher или NodeJS сервер, додека пак Laravel од друга страна користи драјвери за комуникација и интеграција со посредникот. Со цел да ги демонстрираме можностите на Laravel, ќе развиеме мала и навидум едноставна Chat апликација за допишување. Корисниците ќе можат да се допишуваат меѓу себе на јавниот канал, но и да си праќаат приватни пораки. За развој на оваа апликација ќе се послужам со Pusher драјвер и Laravel Echo JavaScript библиотека за воспоставување на врска со Pusher сервисот.

Вовед

Развиена е мала апликација за допишување меѓу корисниците која во иднина треба да се интегрира и да стане дел од веќе готов систем кој е во продукција. За да ја користи апликацијата корисникот мора да биде најавен. Корисничкиот интерфејс е интуитивен и поделен во неколку дела. Од левата страна се наѓа листа од моментално најавените корисници, нивната слика, име и електронска пошта. Во централниот дел е сместен интерфејс во кој се испишуваат пораките. Пораките што ги праќа корисникот се позиционирани десно, а останатите пораки се наоѓаат лево. Во долниот дел се наоѓа текстуална компонента и копче за пишување и праќање на пораки (Слика 1).



Слика 1 - Преглед на корисничкиот интерфејс

Корисникот има на располагање два типа на пораки, јавни и приватни. Јавните пораки се видливи за сите корисници и тие најчесто се користат за групно допишување. Доколку пораката започне со „##“ (на пр. ##user123@email.com) тогаш станува збор за приватна порака која ќе биде видлива само за корисникот за кој е наменета и истата ќе биде обоена во различна боја.

За развој на оваа апликација ќе останам фокусиран и ќе се држам до документацијата на Laravel која е досапна онлајн, за која со сигурност можам да кажам дека

е една од подобрите страни на оваа рамка за развој. Најпрвин ќе ги објаснам основните концепти и поими, а потоа во детали ќе ги анализирам и опишам елементите на Event Broadcasting системот. Тука ќе ги објаснам само најкруцијалните делови од документацијата. За повеќе детали, особено во делот на конфигурација и инсталација, треба да се погледне официјалната документација.

WebSocket протокол

WebSocket [1]

WebSocket е комуникациски протокол кој овозможува full duplex канал преку TCP конекција. Протоколот е стандардизиран од страна на IETF, додека API-то е стандардизирано од страна на W3C.

Првичната замисла на овој протокол е истиот да биде имплементиран од страна на прелистувачите и Web серверите, но може да се користи од било кој клиент или сервер. Протоколот е независен и е базиран врз TCP слојот. Единствена врска со HTTP протоколот е handshake-от кој се праќа како Upgrade барање до HTTP серверот. Главната намена на овој протокол е да овозможи поголема интеракција помеѓу прелистувачот и серверот со трансфер на податоци во реално време. Ова е овозможено преку стандардизиран начин на праќање на пораки до серверот и обратно при што конекцијата останува оторена се додека експлицитно не се побара истата да биде затворена.

За разлика од TCP слојот кој работи со поток од бајти, WebSocket слојот додава ново ниво на апстракција од пораки, односно наместо со бајти работи со концептот порака како основна единица. Дефинира и ws и wss URI шеми за неенкриптирани и енкриптирани конекции соодветно.

WebSocket во Laravel 5.4

Кај многу модерни веб апликации, WebSocket протоколот се користи за да се имплементира кориснички интефејст кој би се менувал во реално време. Најчесто кога ќе настане некоја промена на серверот која е од корист за клиентот, серверот праќа порака преку WebSocket и клиентот соодветно ја прима и ја обработува. Ова е подобро и поробустно решение во однос на постојан pooling на серверот за можни промени.

Секоја промена се дефинира како евент, односно настан. Laravel рамката за развој нуди лесен и едноставен начин за емитирање на настани преку WebSocket конекција. Самиот концепт на настани не ништо друго туку имплементација на „Observer“ шаблонот за развој.

Бидејќи Apache нема нативна подршка за WebSocket, Laravel нуди подршка за исиот преку посредник односно proxy. Laravel располага со неколку драјвери:

- Pusher – драјвер за интеграција со Pusher.com посредник
- Redis – драјвер за интеграција со Redis, кој понатаму би се користел во комбинација со Socket.IO и NodeJS сервер
- Log – драјвер за развивање и дебагирање

Pusher.com

Апликацијата за допишување го користи Pusher драјверот за интеграција. Pusher е хостиран сервис со свој API кој овозможува брза и лесна имплементација и интеграција на двонасочна комуникација кај Web и мобилни апликации. [2] Располага со библиотеки за интеграција со повеќе познати програмски јазици како PHP, Ruby, Python, Java, .NET, Go и Node на серверски дел и JavaScript, Objective-C (iOS) and Java (Android) на клиентски дел. [3]

Со цел да се користи сервисот на Pusher, најпрвин треба да се креира корисничка сметка. По успешно креирање на корисничката сметка, треба да се дефинира апликација. Секоја апликации има уникатен id, key и secret. Една од моќните алакти која стои на располагање е „Debug Console“ (Слика 2). Со помош на неа може во реално време да се следат конекциите, каналите, пораките што се праќаат по нив како и рачно дефинирање и праќање на пораки. Освен тоа на располагање има и статистики во реално време за активните конекции, испратените пораки и слично.

Type	Socket	Details	Time
		Show event creator	
VACATED		Channel: administrator	06:19:07
DISCONNECTION	4828.134842621	Channels: administrator, Lifetime: 22.536108034s	06:19:07
OCCUPIED		Channel: administrator	06:18:45
SUBSCRIBED	4828.134842621	Channel: administrator	06:18:45
CONNECTION	4828.134842621	Origin: http://localhost:8000	06:18:44
VACATED		Channel: administrator	06:18:43
DISCONNECTION	4828.134840943	Channels: administrator, Lifetime:	06:18:43

Слика 2 - Pusher Debug Console

Анализа на Event Broadcasting во Laravel

Настани и канали [4]

Laravel Event Broadcasting системот овозможува емитирање на серверски настани до клиентската JavaScript апликација преку WebSocket со користење на драјвери.

Настаните се емитираат преку канали кои можат да бидат приватни или јавни. Секој корисник може да се претплати на јавен канал без притоа да се најави или авторизира. За да се претплати на било кој од приватните канали, корисникот мора да биде најавен и авторизиран, односно да ги има соодветните привилегии.

Настаните се дефинираат со наредбата php artisan event:generate или пак рачно со креирање на класа која ќе го имплементира ShouldBroadcast интерфејсот. Важно е да се напомене дека доколку рачно се дефинираат настаните, класата треба да ги содржи InteractsWithSockets и SerializesModels trait-от. Интерфејсот наметнува да се имплементира методот broadcastOn(). Овој метод е задолжен да врати каналот на кој што треба да се емитра пораката (Слика 3). Каналот може да биде инстаца од Channel, PrivateChannel или PresenceChannel.

```php
public function broadcastOn()
{
    return new PrivateChannel('order.'.$this->update->order_id);
}
```

Слика 3 - Креирање приватен канал за настан

Доколку експлицитно не се наведе името на настанот во методот broadcastAs(), името на настанот е името на самата класа.

Кога настанот се емитира, сите негови public својства автоматски се сериализираат и се праќаат како payload на самиот настан. За поголема контрола врз тоа кои податоци ќе се емитираат, треба да се имплементира методот broadcastWith кој враќа низа од податоци кои ќе влезат во payload-от.

За доделување привилегии на корисникот за некој канал се дефинираат правила во датотеката routes/channels.php.

```php
Broadcast::channel('order.{orderId}', function ($user, $orderId) {
    return $user->id === Order::findOrNew($orderId)->user_id;
});
```

Слика 4 - Привилегии за претплата на приватен канал

Методот channel() прима два аргумента (Слика 4). Првиот аргумент е име на каналот (вклучувајќи и wildcards). Вториот е callback функција која враќа true или false во зависност од тоа дали корисникот смее да се претплати на каналот или пак не. Првиот аргумент на callback функцијата е моментално најавениот корисник, а вториот е wildcard вредноста од име на каналот доколку истиот е дефиниран и постои во името.

Емитирање на настани

Емитирање на настанот се прави со повикува на функцијата event() на која и се проследува инстанца од настанот како аргумент. Функцијата го емитира настанот до сите претплатени корисници. Слично на event(), може да се користи и функцијата broadcast() (Слика 5).

```
broadcast(new ShippingStatusUpdated($update));
```

Слика 5 - Емитирање настан

Многу често се јавува потребата да се емитираме настанот до сите корисници на каналот, освен моментално најавениот корисник. За таа цел се користи функцијата broadcast() која потоа овозможува надоврзување со методот toOthers() (Слика 6).

```
broadcast(new ShippingStatusUpdated($update))->toOthers();
```

Слика 6 - Емитирање на настан до останатите корисници

За да се имплементира оваа функционалност со помош на Laravel Echo, потребно е барањето што се праќа да го содржи хедерот X-Socket-ID со ID на WebSocket конекцијата од Laravel Echo библотеката. Laravel го чита ID на конекцијата од хедерот и му дава инструкции на broadcaster сервисот да не ги емитира настани на конекциите со соодветниот ID. На тој начин, настанот ќе биде емитиран само до останатите корисници.

Laravel Echo JavaScript библиотека [4]

Laravel Echo е JavaScript библотека која овозможува лесен и едноставен начин за претплатување на каналите и слушање за можни настани емитирани од Laravel. Библотеката е достапна преку NPM менаџерот за пакети. Бидејќи во овој проект користам Pusher драјвер, потребно е да се инсталира и pusher-js пакетот.

Креирањето на Echo инстанца е тривијално. Најпрвин се импортира модулот, а потоа се инстанцира со соодветните параметри за Pusher сервисот (Слика 7).

```
import Echo from "laravel-echo"

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: 'your-pusher-key'
});
```

Слика 7 - Креирање на инстанца

За претплатување на некој од каналите, првин се повикува channel() методот за да се добие инстанцата за соодветниот канал, а потоа се повикува listen() методот за да слуша и очекува настанот со име проследен како аргумент (Слика 8).

```
Echo.channel('orders')
    .listen('OrderShipped', (e) => {
        console.log(e.order.name);
    });
```

Слика 8 - Претплатување на канал и слушање настани

За претплатување на приватен канал, се користи методата private(). Доколку има потреба за слушање на повеќе настани на еден ист канал, методот listen() може да се повика неколку пати последователно. За одјавување од некој канал, едноставно се повикува методата leave().

Канали за присутност [4]

Каналите за присутност се изградени врз основата на приватните канали нудејќи додадтни функционалности како што е информацијата за моментално претплатените корисници на соодветниот канал. Со помош на каналите за присутност, може брзо и лесно да се креираат колаборативни функционалности како на пример известување на корисникот кога некој од останатите корисници ја гледа истата страна.

Сите канали за присутност се и приватни канали. Тоа значи дека корисникот мора да биде најавен и авторизиран. Меѓутоа, кога се дефинира callback функцијата, наместо да врати true, треба да врати низа од податоци за корисникот (Слика 9). Низата со податоци ќе биде достапна на претплатените корисиници на каналот за присутност. Слично како кај приватните канали, доколку корисникот ги нема соодветните привилегии за да се претплати на каналот за присутност, callback функцијата треба да врати false.

```
Broadcast::channel('chat.*', function ($user, $roomId) {
    if ($user->canJoinRoom($roomId)) {
        return ['id' => $user->id, 'name' => $user->name];
    }
});
```

Слика 9 - Callback функција на канал за присутност

За претплата на некој од каналите за присутност, се користи join() методата на Laravel Echo библиотеката. Оваа метода враќа PresenceChannel имплементација, која освен стандардната listen() метода, ги има и методите here(), joining() и leaving (Слика 10).

```
Echo.join(`chat.${roomId}`)
    .here((users) => {
        //
    })
    .joining((user) => {
        console.log(user.name);
    })
    .leaving((user) => {
        console.log(user.name);
    });
```

Слика 10 - Претплата на канал за присутност кај Laravel Echo

Callback функцијата проследена на here() методот ќе се изврши веднаш по успешно претплатување на каналот и ќе прими низа од податоци која ги содржи информациите за сите корисници кои се моментално претплатени на овој канал. Callback функцијата на методота joining() ќе се изврши секој пат кога некој нов корисник ќе се претплати на каналот, додека пак callback-от на leaving() методата ќе се изврши кога некој од корисниците ќе се одјави од каналот.

Исто како и кај приватните и јавни канали, и кај каналите за присутност може да се емитираат настани, со помош на broadcast() и toOthers() соодветно. Методот broadcastOn() треба да врати инстанца од PresenceChannel() класата. На клиентскиот дел, исто така може да се слуша за настани од интерес со методата listen() (Слика 11).

```
Echo.join(`chat.${roomId}`)
    .here(...)
    .joining(...)
    .leaving(...)
    .listen('NewMessage', (e) => {
        //
    });
```

Слика 11 - Слушање настани кај канали за присутност

Клиентски настани

Понекогаш се јавува потребата за емитирање на настан до другите претплатени клиенти без притоа да се прави барање до Laravel апликацијата. Пример за ваков настан е известувањето дека некој од корисниците „пишува во моментот ...“ слично како кај Facebook и Skype. За емитирање на ваков тип настани, се користи методата whisper() од Laravel Echo библиотеката (Слика 12).

```
Echo.channel('chat')
    .whisper('typing', {
        name: this.user.name
    });
```

Слика 12 - Клиентски настани

За слушање на клиентски настани, се користи listenForWhisper() медодата (Слика 13).

```
Echo.channel('chat')
    .listenForWhisper('typing', (e) => {
        console.log(e.name);
    });
```

Слика 13 - Слушање клиентски настани

Анализа на апликација за допишување

Клонирање на GitHub репозиториумот

Проектот е достапен јавно на GitHub - https://github.com/vladicaku/laravel-websocket-simple-chat-application. За да го клонираме репозиториумот ја извршуваме наредбата git clone https://github.com/vladicaku/laravel-websocket-simple-chat-application.git

Превземање на потребните библиотеки

Со цел да ги превземеме потребните библиотеки кои што ги користи нашиот Laravel проект, ја извршуваме наредбата composer install. По успешното превземање на библиотеките, ја извршуваме наредбата php artisan key:generate.

Превземеме на потребните Node модули кои се користат на front-end делот го вршиме со наредбата npm install.

Конфигурација на .env

Најпрвин треба да се постават параметрите за конекцијата со базата, како што е корисничкото име, лозинка и името на базата во зависност од вашиот систем на кој што ќе го стартувате проектот.

Параметарот BROADCAST_DRIVER треба да ја има вредност „pusher“. Со тоа наведуваме дека ќе го користиме Pusher драјверот за Event Broadcasting системот. Потоа треба да се постават параметрите PUSHER_APP_ID, PUSHER_APP_KEY и PUSHER_APP_SECRET според оние на вашата Pusher инстанца.

Конфигурација на систем за најава и иницијализација на база [5]

Laravel 5.4 ни овозможува да го конфигурираме системот за најава со помош на неколку наредби. Најпрвин треба да ја извршиме наредбата php artisan make:auth која ќе ги постави соодветните рути и контролери. Исто така ќе генерира и страници за најава и за регистрирање кои ќе бидат заосновани на Vue.js библиотеката.

Според спецификациите наведени во документацијата за Laravel Authentication, треба да креираме табела „users“ во која ќе ги чуваме корисниците. За да ја креираме табелата, ќе ги извршиме веќе дефинираните миграции во нашиот проект со помош на наредбата php artisan migrate. Миграциите ќе ги креираат табелите „users“, „password_resets“ и „migrations“.

За да додадеме иницијални корисници во нашата база, ќе ја извршиме наредбата php artisan db:seed --class=UsersTableSeeder која ќе го активира соодветниот Seeder и во базата ќе ги додаде корисниците Mladen Jovanovski и Vladica Jovanovski. Алтернативен начин за додавање на корисници е преку страната за регистрирање која ја генериравме погоре.

Конфигурација на Webpack

Поради bug во конфигурацијата на моменталната верзија на Laravel Mix, ќе конфигурираме засебна инстанца на Webpack. Конфигурацијата е сместена во датотеката webpack.config.js. Скриптите и зависностите кои што ќе ги користиме се дефинирани во package.json датотеката.

Дефинирање на настани

Нашата апликација има еден приватен канал кој е достапен на сите најавени корисници. На овој канал корисниците ќе можат да се допишуваат меѓу себе и пораките кои што ќе ги праќаат ќе бидат јавни достапни и видливи за сите корисници. Настанот за праќање на јавни порараки е PublicMessageEvent (Слика 14), а името на каналот е „public“.

```
class PublicMessageEvent implements ShouldBroadcast {

    use InteractsWithSockets, SerializesModels;

    public $data;

    public function __construct($data) {
        $this->data = $data;
    }


    public function broadcastOn() {
        return new PresenceChannel("public");
    }
}
```

Слика 14 - Настан за јавни пораки

Како што може да се забележи од Слика 14, „public“ каналот е канал за присутност. Како што споменавме на почетокот, секој канал за присутност е всушност и приватен канал. Каналите за присутност ни даваат информација за корисници кои се претплатени на каналот во моментот. Во датотеката channels.php е дефинирана callback функција која враќа низата со податоци за корисникот (Слика 15).

```php
Broadcast::channel('public', function ($user) {
    if (!Request::session()->has("avatar")) {
        Request::session()->put("avatar", 'avatar' . rand(1, 7));
    }

    return [
        'id' => $user->id,
        'name' => $user->name,
        'email' => $user->email,
        'avatar' => Request::session()->get("avatar"),
    ];
});
```

Слика 15 - Информации за корисникот за каналот за присутност

За праќање на приватни пораки помеѓу корисниците кои што нема да бидат јавно достапни се користи настанот PrivateMessageEvent (Слика 16). Секој најавен корисник освен што е претплатен на „public“ каналот, исто така е претплатен и на каналот „user-ID“, каде што ID е конкретно ID на корисникот. Тоа значи дека секој корисник има посебен канал само за него преку кој ќе му бидат доставени приватните пораки.

```php
class PrivateMessageEvent implements ShouldBroadcast {

    use InteractsWithSockets, SerializesModels;

    public $data;

    public function __construct($data) {
        $this->data = $data;
    }


    public function broadcastOn() {
        return new PrivateChannel('user-' . $this->data->to);
    }
}
```

Слика 16 - Настан за приватни пораки

Во датотеката channels.php е дефинирана callback функција со која се дозволува или забранува пристапот на корисникот до соодветниот приватен канал (Слика 17).

```php
Broadcast::channel('user-{userId}', function ($user, $userId) {
    return $user->id === $userId;
});
```

Слика 17 - Канал за приватни пораки

Праќање пораки – back end

Има единствен контролер кој е одговорен за прикажување на страната со апликацијата за допишување и процесирање на пораките. Барањата до овој контролер поминуваат низ auth middleware со цел да се ограничи пристапот само за најавени корисници. Методата index() е мапирана на „/chat“ рутата и служи за прикажување на страната (Слика 18).

```php
public function index(Request $request) {
    $user = $this->convertUser($request);
    return view('chat.index', compact('user'));
}
```

Слика 18 - Index метода

До страната „char/index.blade.php“ се проследува објект со информации за моментално најавениот корисник кој се креира со помош на методата convertUser(). Оваа метода го зема најавениот корисник со помош на Auth фасадата и ги сместува податоците во празен објект заедно со случајно генерира слика (аватар) (Слика 19).

```php
public function convertUser(Request $request) {
    $user = Auth::user();
    if (!$request->session()->has("avatar")) {
        $request->session()->put("avatar", 'avatar' . rand(1, 7));
    }

    $model = new \stdClass();
    $model->id = $user->id;
    $model->name = $user->name;
    $model->email = $user->email;
    $model->avatar = $request->session()->get("avatar");

    return $model;
}
```

Слика 19 - Метода за генерирање објект со информации за најавениот корисник

Методата sendMessage() е мапирана на „/chat/send-message“ рутатата во web.php датотеката. Кога корисникот праќа пораки тие се проследуваат преку Ajax барање до оваа рута. Методата го парсира барањето и ја очекува пораката како „message“ параметар. Доколку пораката започнува со „##“ (на пр. ##user123@email.com) тогаш станува збор за приватна порака и истата треба да биде проследена до приватниот канал за соодветниот корисник. Во тој случај се емитира PrivateMessageEvent настанот заедно со пораката и корисникот за кој е наменета. Во спротивно, пораката е јавна и истата се праќа преку PublicMessageEvent настанот. (Слика 20)

```php
public function sendMessage(Request $request) {

    $message = $request->input('message', 'Blank message');
    $exploded = explode(' ', trim($message));
    $receiver = $exploded[0];

    $model = [
        'user' => $this->convertUser($request),
        'time' => date('H:i'),
    ];

    if (substr($receiver, 0, 2) === '##') {
        $model['to'] = User::where('email', substr($receiver, 2))->first()->id;
        $model['message'] = substr($message, strlen($receiver));
        broadcast(new PrivateMessageEvent($model))->toOthers();

    } else {
        $model['message'] = $message;
        broadcast(new PublicMessageEvent($model))->toOthers();
    }

    return response()->json([
        'status' => 'ok'
    ]);
}
```

Слика 20 - Метода за емитирање настани

Праќање пораки – front end

Функционалноста на front end делот се наоѓа во датотеката chat.js која е сместена во public/js директориумот. Скриптата користи ES6 синтакса што значи дека треба да се компајлира при секоја промена со помош на веќе конфигурираната инстанца на webpack.

Откако страната успешно ќе се вчита, се конфигурира Laravel Echo инстанца. Потоа се претплатуваме на каналот за присутност и приватниот канал и слушаме за соодветни настани. За секој од настани се повикува функција која е задолжен да го обработи настанот и да го ажурира корисничкиот интерфејс (Слика 21).

```javascript
/* Join public channel */
window.Echo.join('public')
    .here((users) => {
        $('.user').remove();
        users.forEach( u => joining(u));
    })
    .joining((user) => {
        joining(user);
    })
    .leaving((user) => {
        leaving(user);
    }).listen('PublicMessageEvent', (e) => {
    messageReceived(e.data);
});

/* Join my private channel */
window.Echo.private('user-' + me.id).listen('PrivateMessageEvent', (e) => {
    messageReceived(e.data, true);
});
```

Слика 21 - Претплата на канали

Конечно, при клик на копчето „Send“ или со притискање на „Enter“ се повикува функцијата sendMessage() која креира Ajax повик до рутата „/chat/send-message“ и ги потполнува потребните хедери (Слика 22).

```
$.ajax({
    url: "/chat/send-message",
    type: "get",
    data: {
        'message': message
    },
    beforeSend: function (request) {
        request.setRequestHeader("X-Socket-ID", window.Echo.socketId());
    },
    success: function (response) {
        $(element).appendTo(chatDiv);
        chatDiv.scrollTop(chatDiv.prop("scrollHeight"));
    },
    error: function (xhr) {
    }
});
```

Слика 22 - Ajax повик

Заклучок

Непостоењето на нативна поддршка за WebSocket кај Apache серверот не претставуваше голем проблем при развој на апликацијата за допишување. Тимот на Laravel се потрудил сите потребни елементи и технологии да ги инкорпорира во една заокружена целина и на нас да ни остави само да продолжиме со развивање на апликацијата, без притоа да губиме време во барање најразлични библиотеки и нивно поврзување. Концептот на Event Broadcasting е извонреден и прилично лесен за употреба. Интеграцијата со Laravel Echo библиотеката ја прави целата приказна уште поинтересна. Модуларноста при изборот на драјвер ни дава поголема слобода при имплементација на решнието за соодветниот проблем. Документацијата е детална и јасна и ми го олесни процесот за развој на апликацијата. Генералниот впечаток што го стекнав при користењето на Event Broadcasting ситемот, но и со целата рамка за развој воопшо, е задоволството и имањето слобода при моделирање на решението по своја желба од една страна, но и постење на ограничувачки фактор и насоки во однос на добрите праткити за равој и изборот на технологии од друга страна.

Литература

[1] “WebSocket,” 2 2017. [Мрежен]. Available: https://en.wikipedia.org/wiki/WebSocket.

[2] “Pusher,” 2 2017. [Мрежен]. Available: https://pusher.com/docs.

[3] “GitHub - Pusher Community,” 2 2017. [Мрежен]. Available: https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html.

[4] “Laravel,” Laravel, 2 2012. [Мрежен]. Available: https://laravel.com/docs/5.4/broadcasting.

[5] “Laravel Authentication,” Laravel, Ffebruary 2017. [Мрежен]. Available: https://laravel.com/docs/5.4/authentication.

Додатоци

Содржина