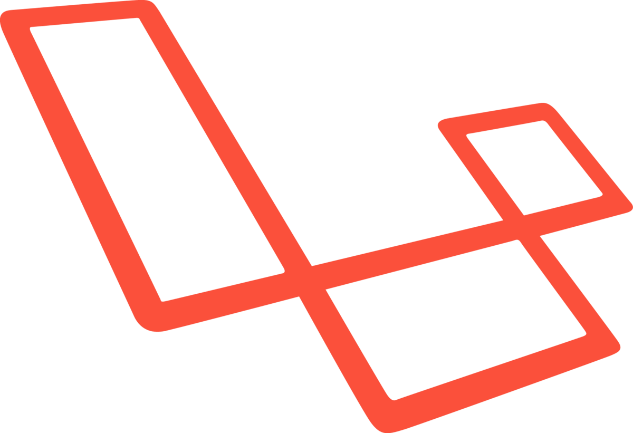
Универзитет „Св. Кирил и Методиј“ – Скопје

Факултет за информатички науки и компјутерско инженерство (ФИНКИ)



Разработка на WebSockets и имплементација во Laravel 5.4

Изработил:

Владица Јовановски 121202

# Абстракт

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. Непосредно, 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 го користиме Pusher драјверот и Laravel Echo JavaScript библиотеката за воспоставување на врска со Pusher сервисот.

# Вовед

За развој на нашата апликација ќе останеме фокусирани и ќе се држиме 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 се запознаеме со основните концепти и поими, а потоа во детали ќе ги разгледаме и опишеме елементите на Event Broadcasting системот. Тука ќе ги објасниме само најкруциалните делови од документацијата. За повеќе детали, особено во делот на конфигурација и инсталација, треба да се повикаме на официјалната документација.

## 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 абстракција од пораки, односно наместо со бајти работи со концептот порака како основна едниница. 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.

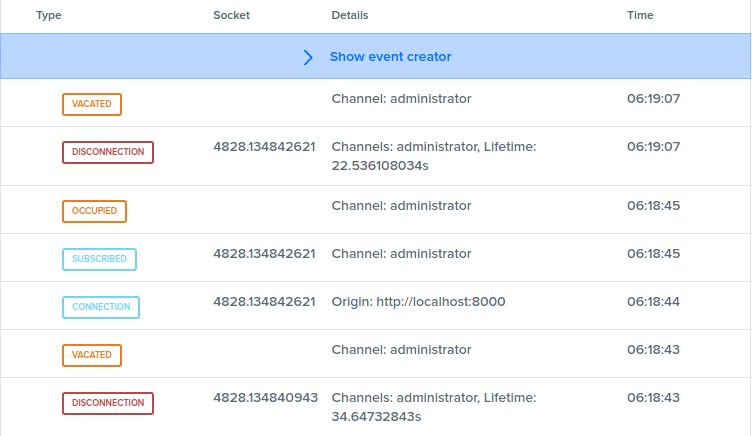
Секоја промена ќе ја дефинираме како евент, односно настан. Laravel рамката за развој ни нуди лесен и едноставен начин да направиме емитирање на нашите настани преку WebSocket конекција. Самиот концепт на настани не ништо друго туку имплементација на „Observer“ шаблонот за развој.

Бидејќи Apache нема нативна подршка за WebSocket, Laravel ни 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

Нашата апликација ќе 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]

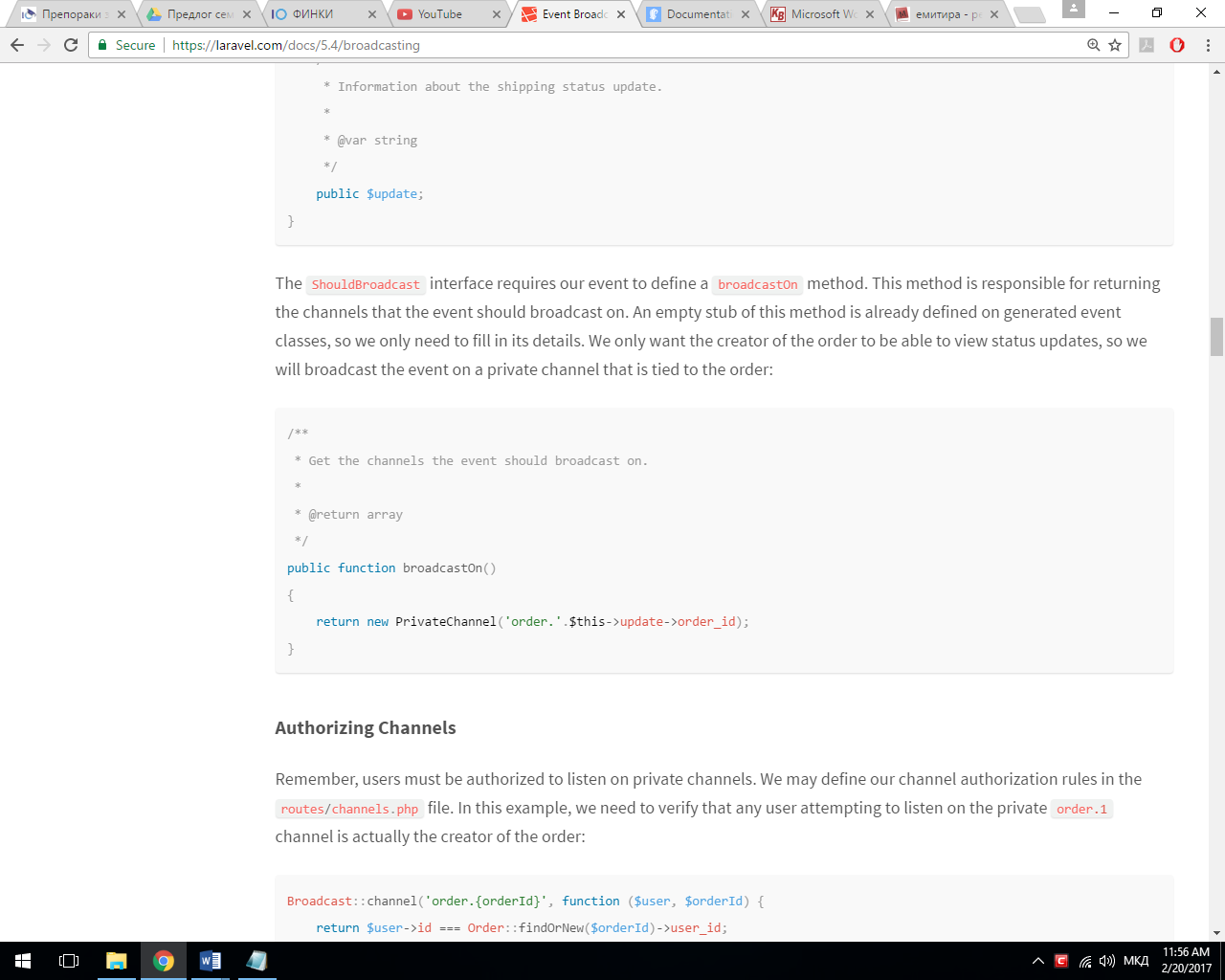
 Со цел да ги користиме сервисот на Pusher треба да креираме корисничка сметка. По успешно креирање на корисничката сметка, треба да дефинираме апликација. Секоја апликации има уникатен id, key и secret. Една од моќните алакти кои ги имаме на располагање е „Debug Console“ (Слика 1). Со помош на неа може во реално време да ги следиме конекциите, каналите и пораките што се праќаат по нив како и рачно да дефинираме и праќаме пораки. Освен тоа на располагање имаме и статистики во реално време за активните конекции, испратените пораки и слично.

Слика 1 - Pusher Debug Console

## Настани и канали [4]

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

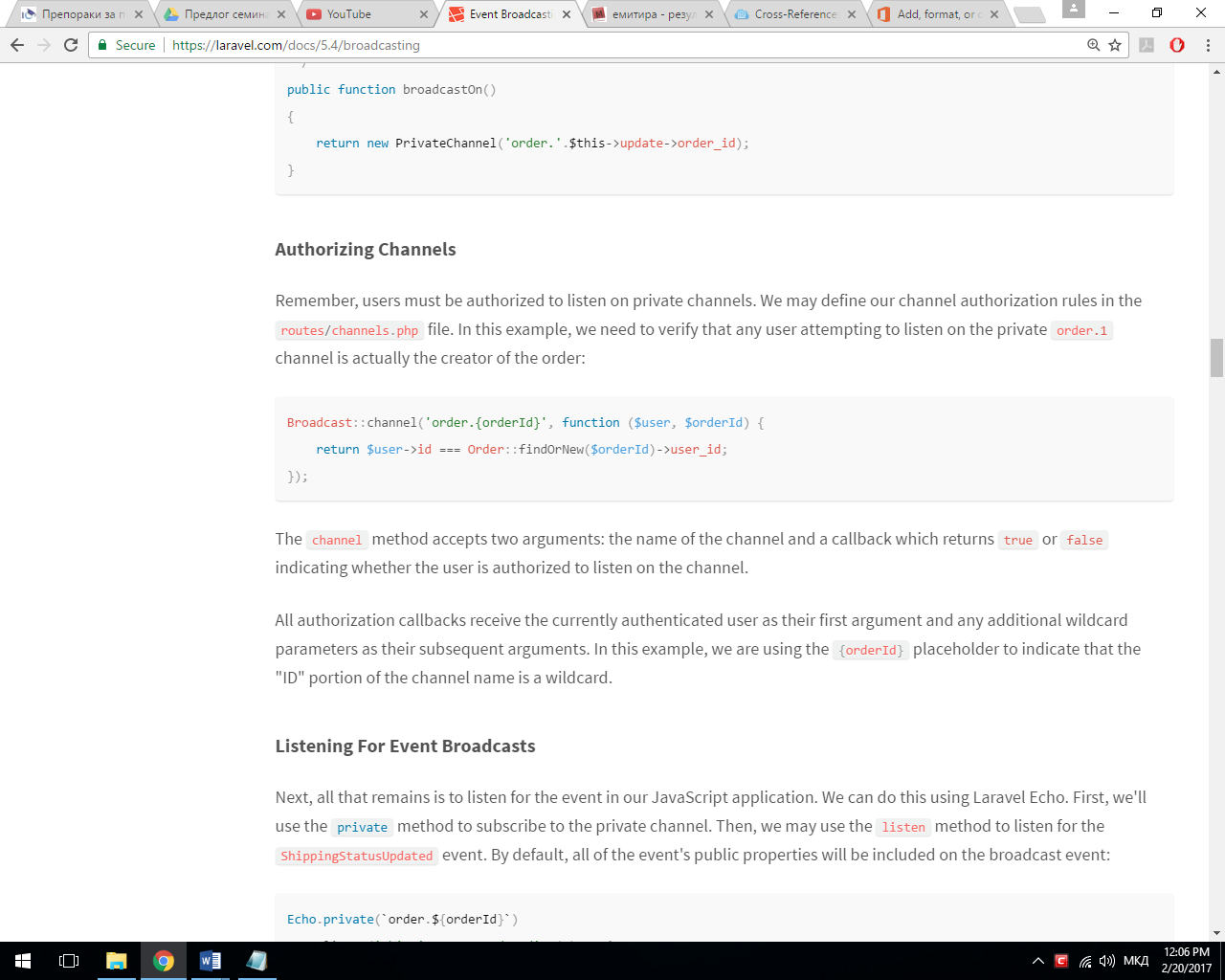
Настаните се емитираат преку канали кои можат да бидат приватни или јавни. Секој корисник може да се претплати на јавен канал без притоа да се најави или ауторизира. За да се претплати на било кој од приватните канали, корисникот мора да биде најавен и ауторизиран, односно да ги има соодветните привилегии.

 Настаните се дефинираат со наредбата php artisan event:generate или пак рачно сo креирање на класа која ќе го имплементира ShouldBroadcast интерфејсот. Важно е да се напомене дека доколку рачно ги дефинираме настаните, класата треба да ги содржи InteractsWithSockets и SerializesModels trait-от. Интерфејсот не обврзува да го имплементира методот broadcastOn(). Овој метод е задолжен да врати каналот на кој што треба да се емитра пораката (Слика 2). Каналот може да биде инстаца од Channel, PrivateChannel или PresenceChannel.

Слика 2 - Креирање приватен канал за настан

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

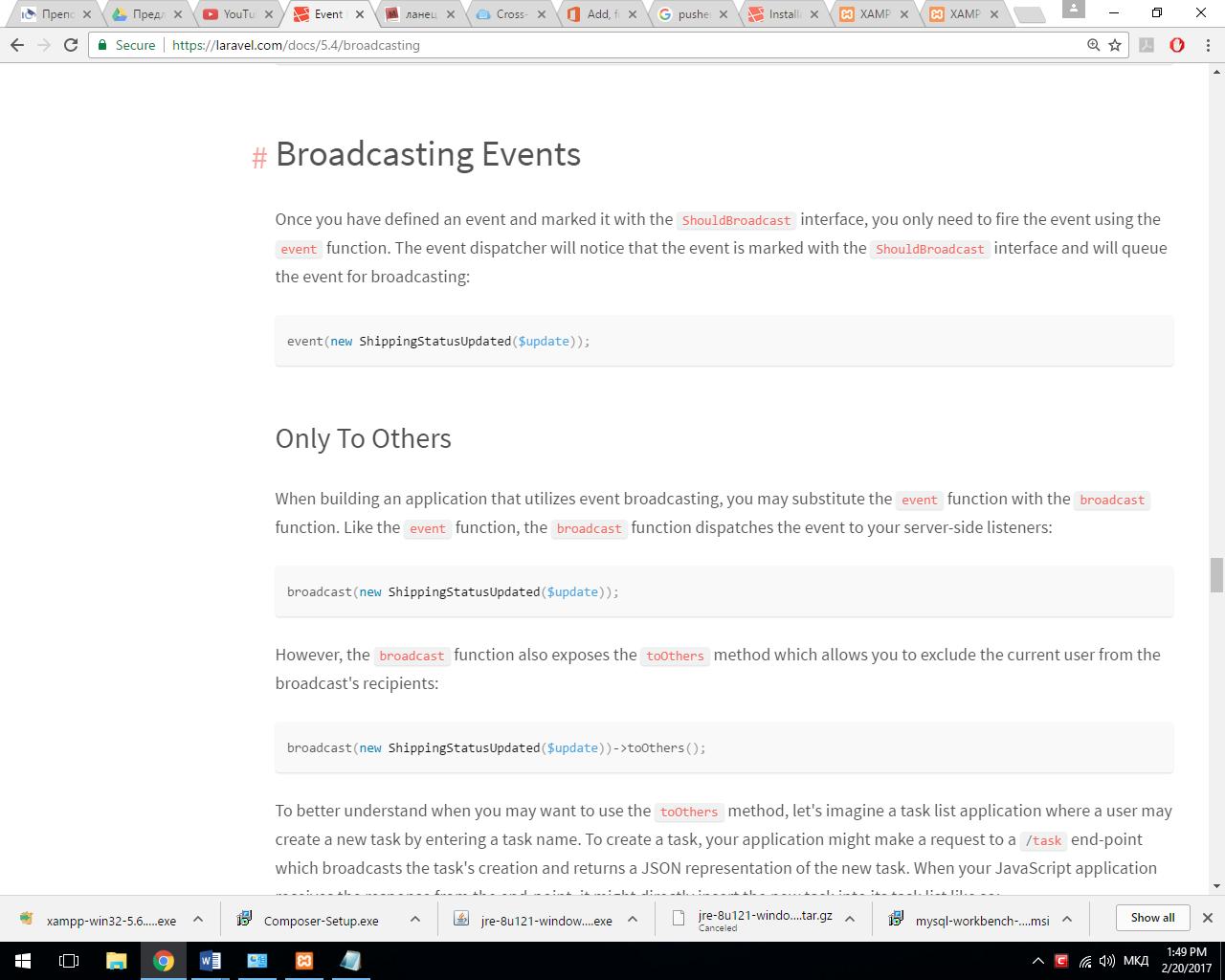
Кога настанот се емитира, сите негови public својства автоматски се сериализираат и се праќаат како payload на самиот настан. Доколку сакаме поголема контрола врз тоа кои податоци ќе се емитираат, треба да го имплементираме методот broadcastWith кој враќа низа од податоци кои ќе влезат во payload-от.

 За да му дадеме привилегии на корисникот да се претплати на приватен канал дефинираме правила во датотеката routes/channels.php.

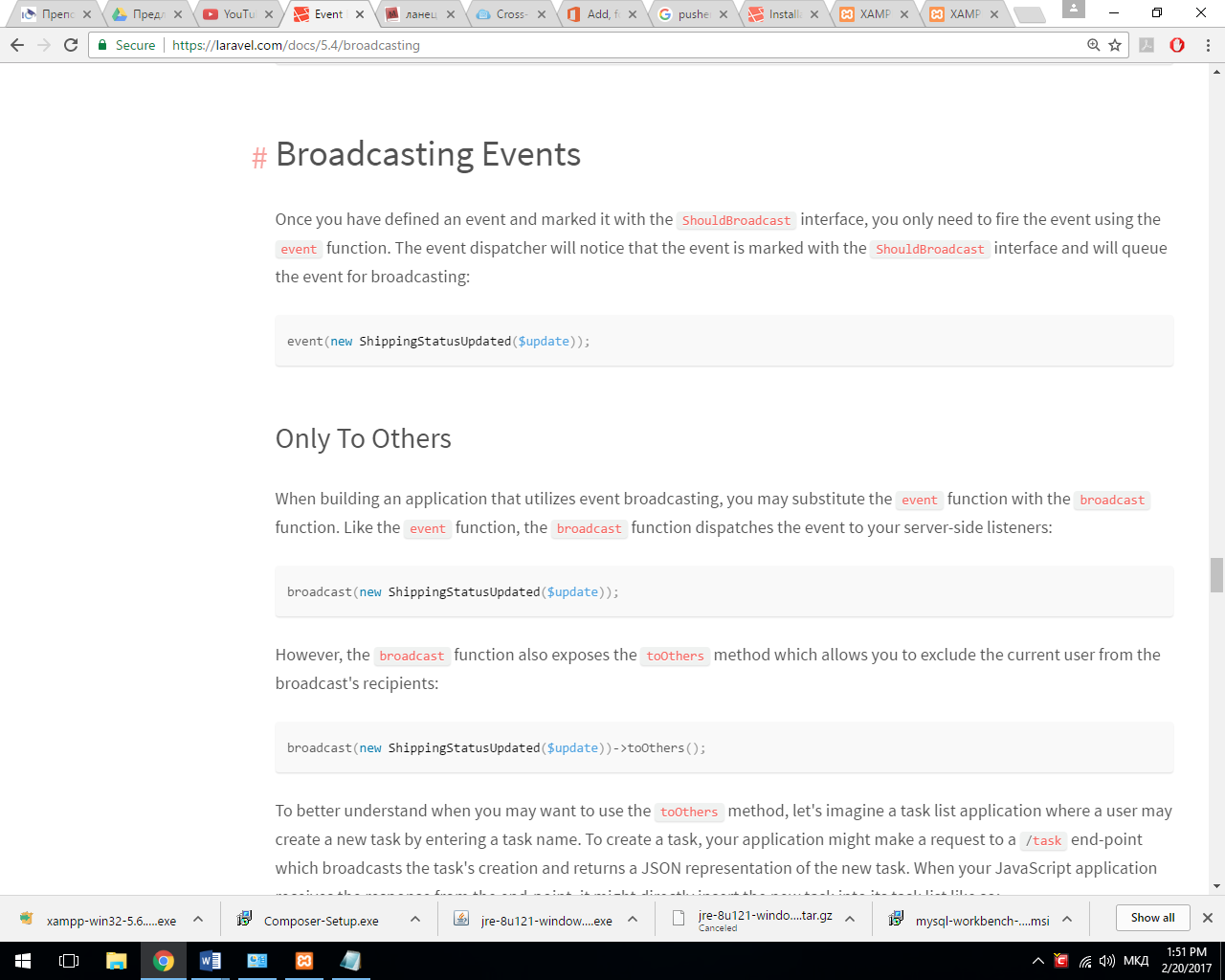
Слика 3 - Привилегии за претплата на приватен канал

Методот channel() прима два аргумента (Слика 3). 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 инстаца од настанот како аргумент. Финкцијата го емитира настанот до сите претплатени корисници. Слично на event(), може да ја користиме и функцијата broadcast() (Слика 4).

Слика 4 - Емитирање настан

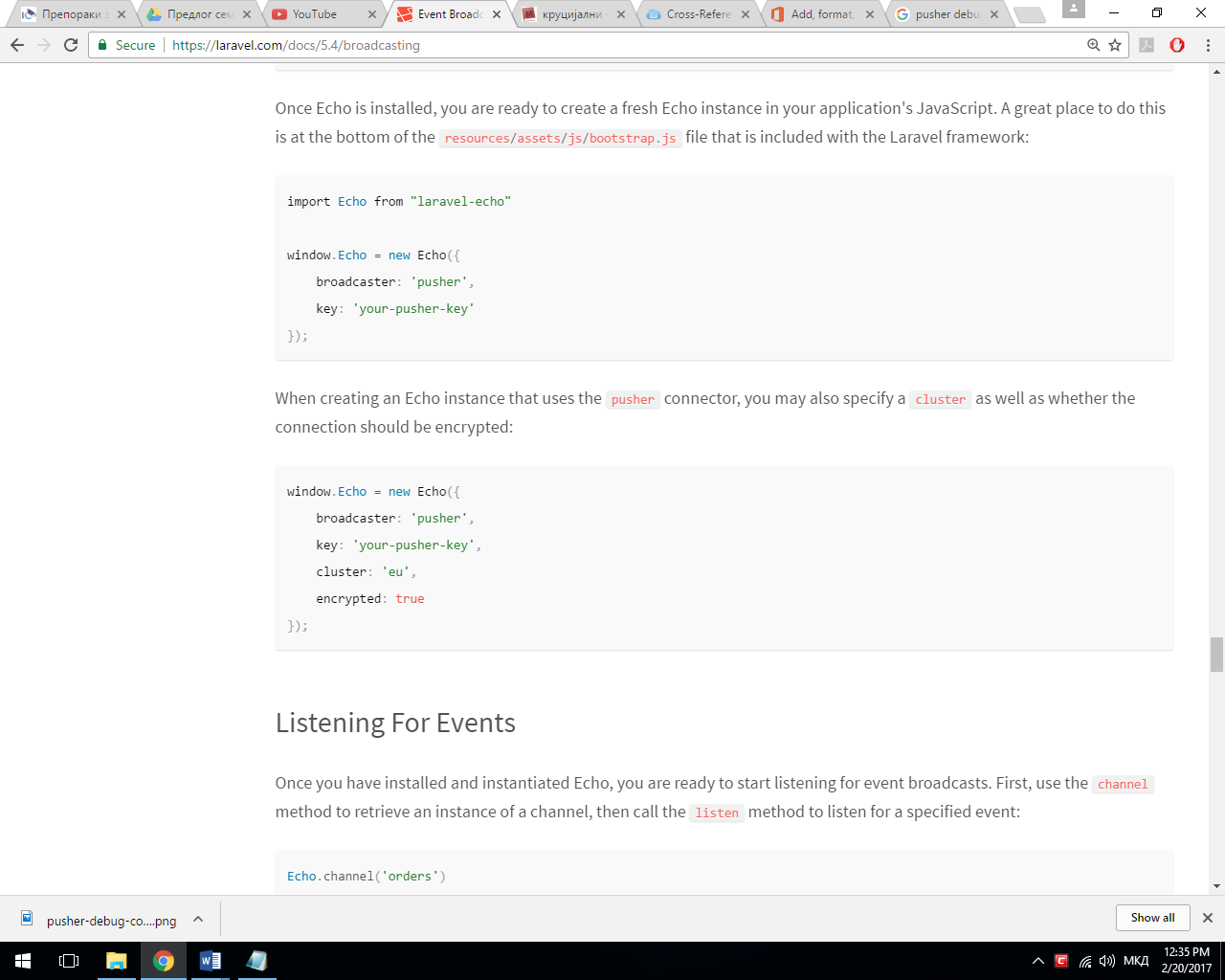
 Многу често имаме потреба да го емитираме настанот до сите корисници на каналот, освен моментално најавениот корисник. За таа цел ке ја повикаме функцијата broadcast() која потоа ни овозможува да се надоврземе со методот toOthers() (Слика 5).

Слика 5 - Емитирање на настан до останатите корисници

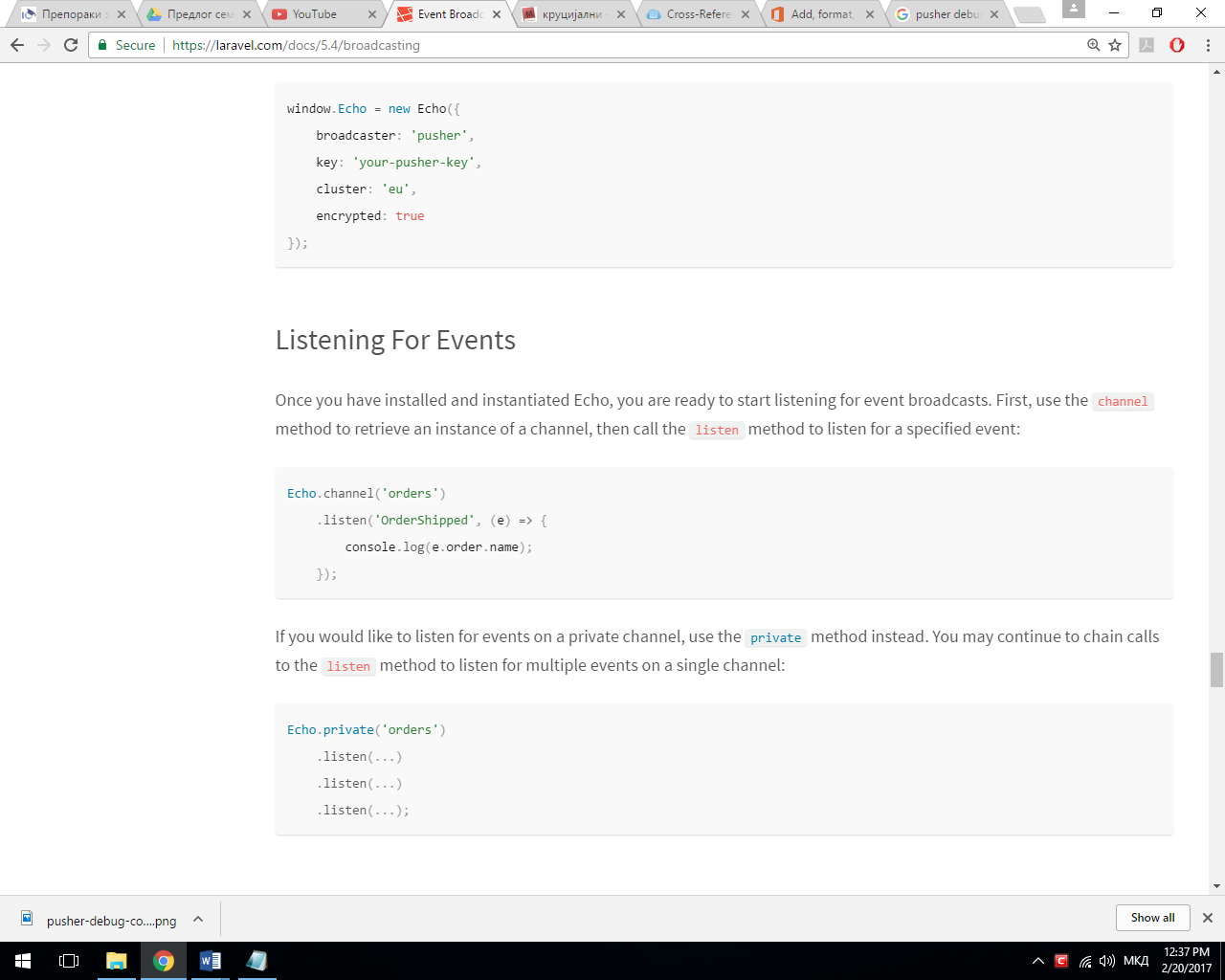
За да се имплементира оваа функционалност со помош на Laravel Echo, потребно е барањето што се праќа да го содржи хедерот X-Socket-ID со ID на WebSocket конекцијат од Laravel Echo библотеката. Laravel го чита ID на конекцијата од хедерот и му дава инструкции на broadcaster сервисот да не ги емитира настани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

## Laravel Echo JavaScript библиотека [4]

Laravel Echo е JavaScript библотека која ни овозможува на лесен начин да се претплатиме на каналите и да слушаме за можни настани емитирани од Laravel. Библотеката е достапна преку NPM пакет менаџерот. Бидејќи во овој проект работиме со Pusher драјверот, потребно е да го инсталираме 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 6).

Слика 6 - Креирање на инстанца

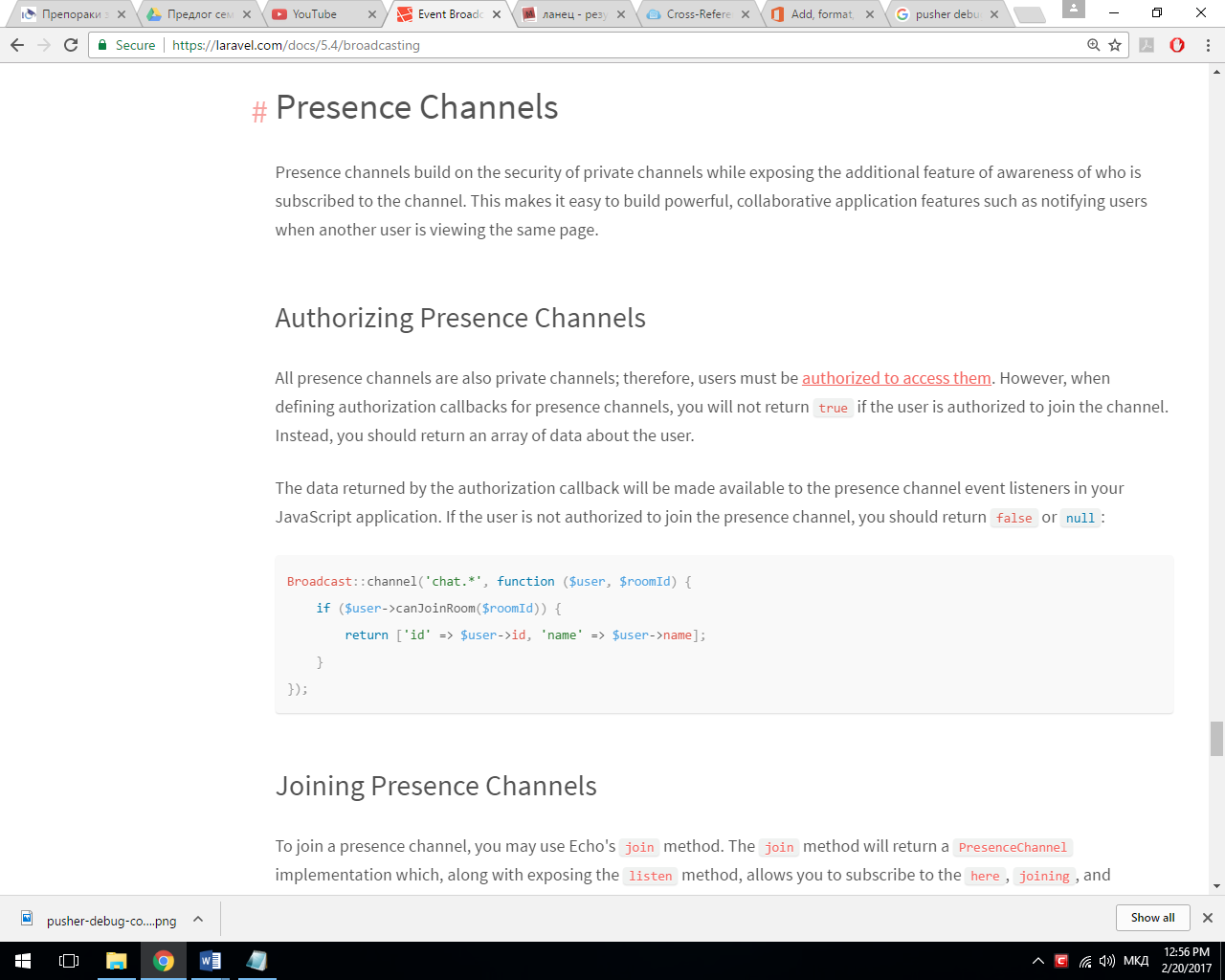
Доколку сакаме да се претплатиме на некој од каналите, го повикуваме channel() методот за да ја добиеме инстанцата за соодветниот канал, а потоа го повикуваме listen() методот за да слушаме за соодветен настан (Слика 7).

Слика 7 - Претплатување на канал и слушање настани

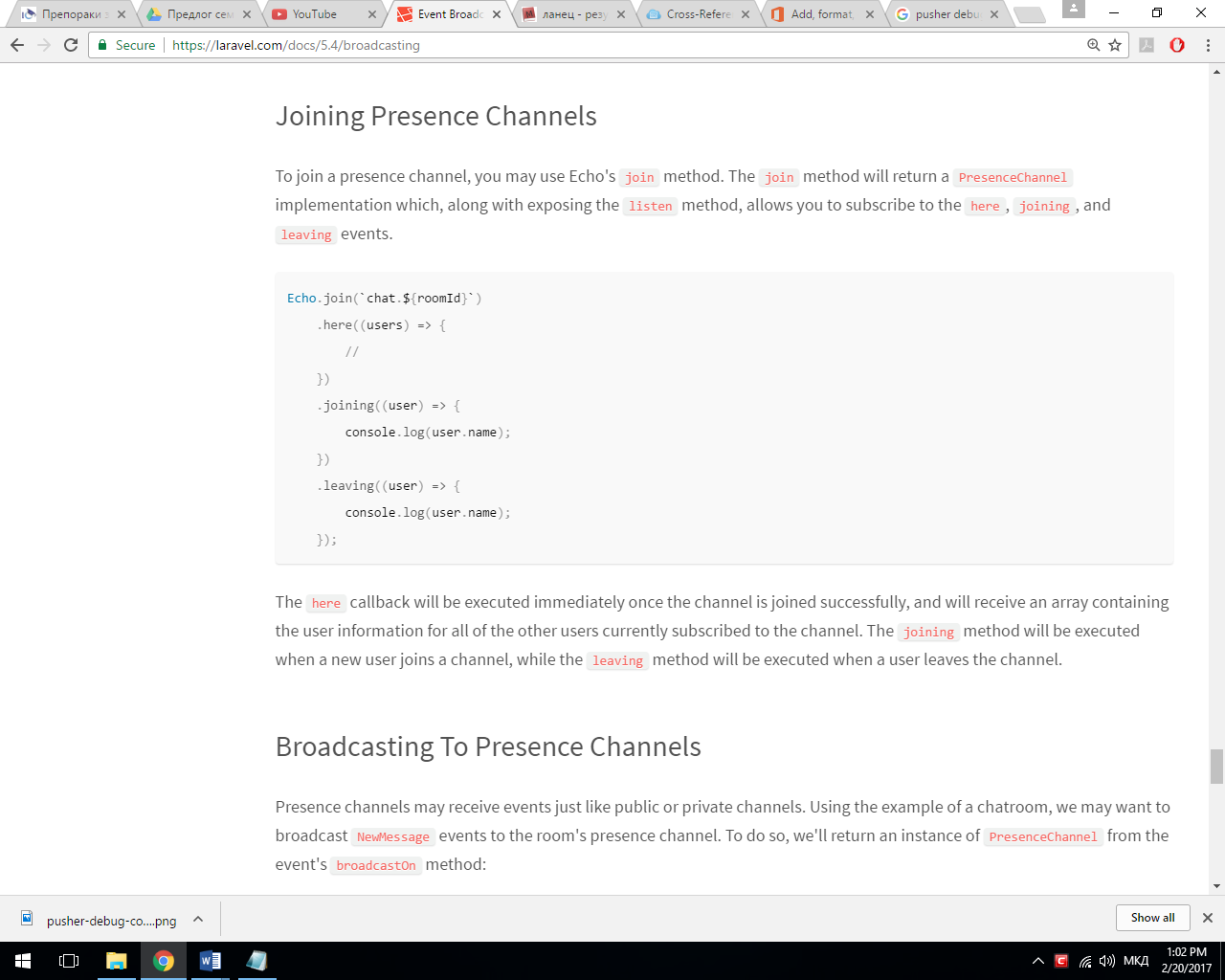
За да се претплатиме на приватен канал, ја повикуваме методата private(). Доколку сакаме да слушаме повеќе настани на еден ист канал, методот listen() може да го повикаме неколку пати последователо. За да се одјавиме од некој канал, едноставно ја користиме методата leave().

## 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 креираме колаборативни функционалности како на пример известување на корисникот кога некој од останатите корисници ја гледа истата страна.

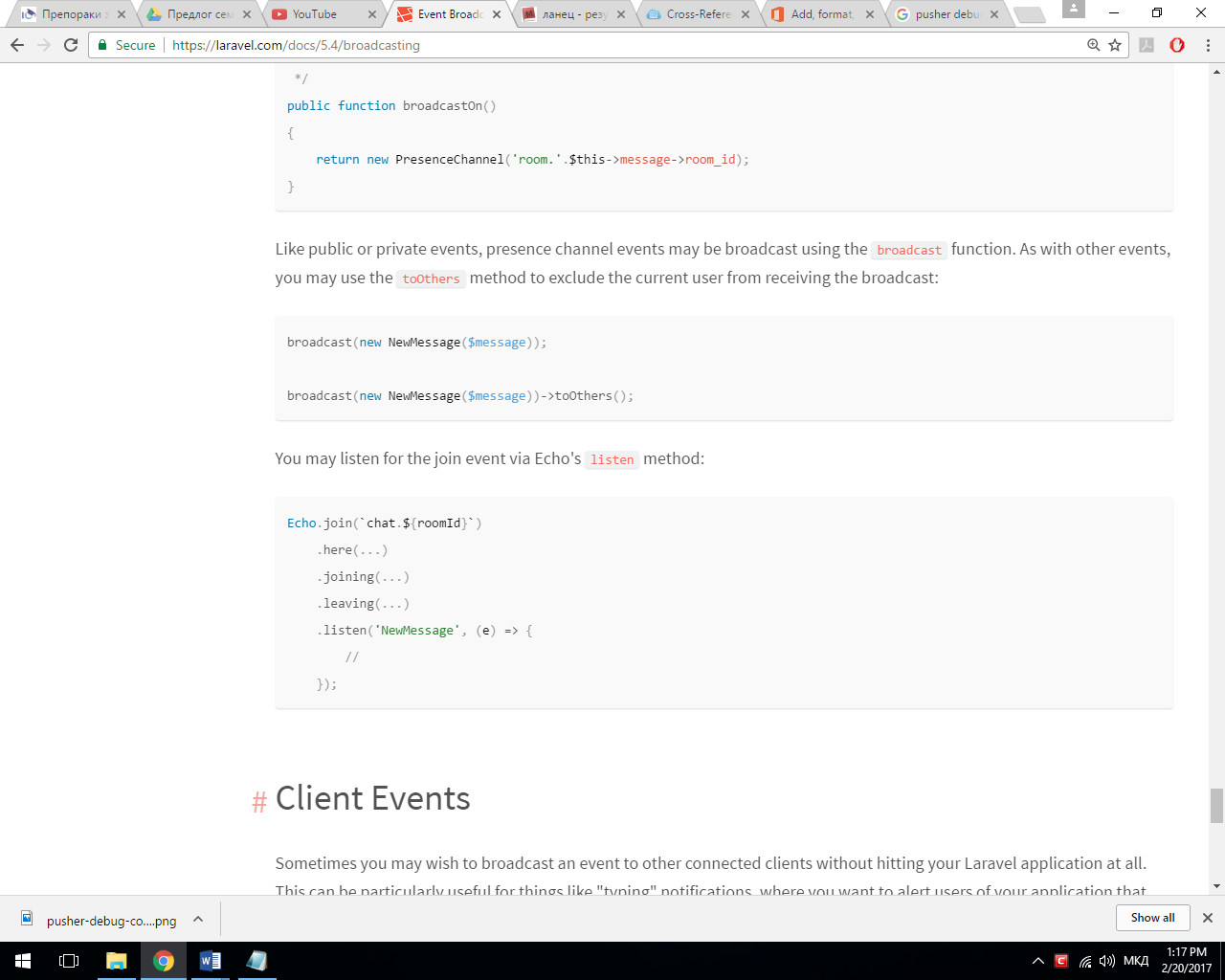
 Сите канали за присутност се и приватни канали. Тоа значи дека корисникот мора да биде најавен и ауторизиран. Меѓутоа, кога ја дефинираме callback функцијата, наместо да вратиме true, треба да вратиме низа од податоци за корисникот (Слика 8). Низата со податоци ќе биде достапна на претплатените корисиници на каналот за присутност. Слично како кај приватните канали, доколку корисникот ги нема соодветните прицилегии за да се претплати на каналот за присутност, callback функцијата треба да врати false.

Слика 8 - Callback функција на канал за присутност

 За да се претплатиме на каналите за присутност, ја користиме join() методата на Laravel Echo библиотеката. Оваа метода враќа PresenceChannel имплементација, која освен стандардната listen() метода, ни овозможува да се претплатиме и на „here“, „joining“ и „leaving“ настаните (Слика 9).

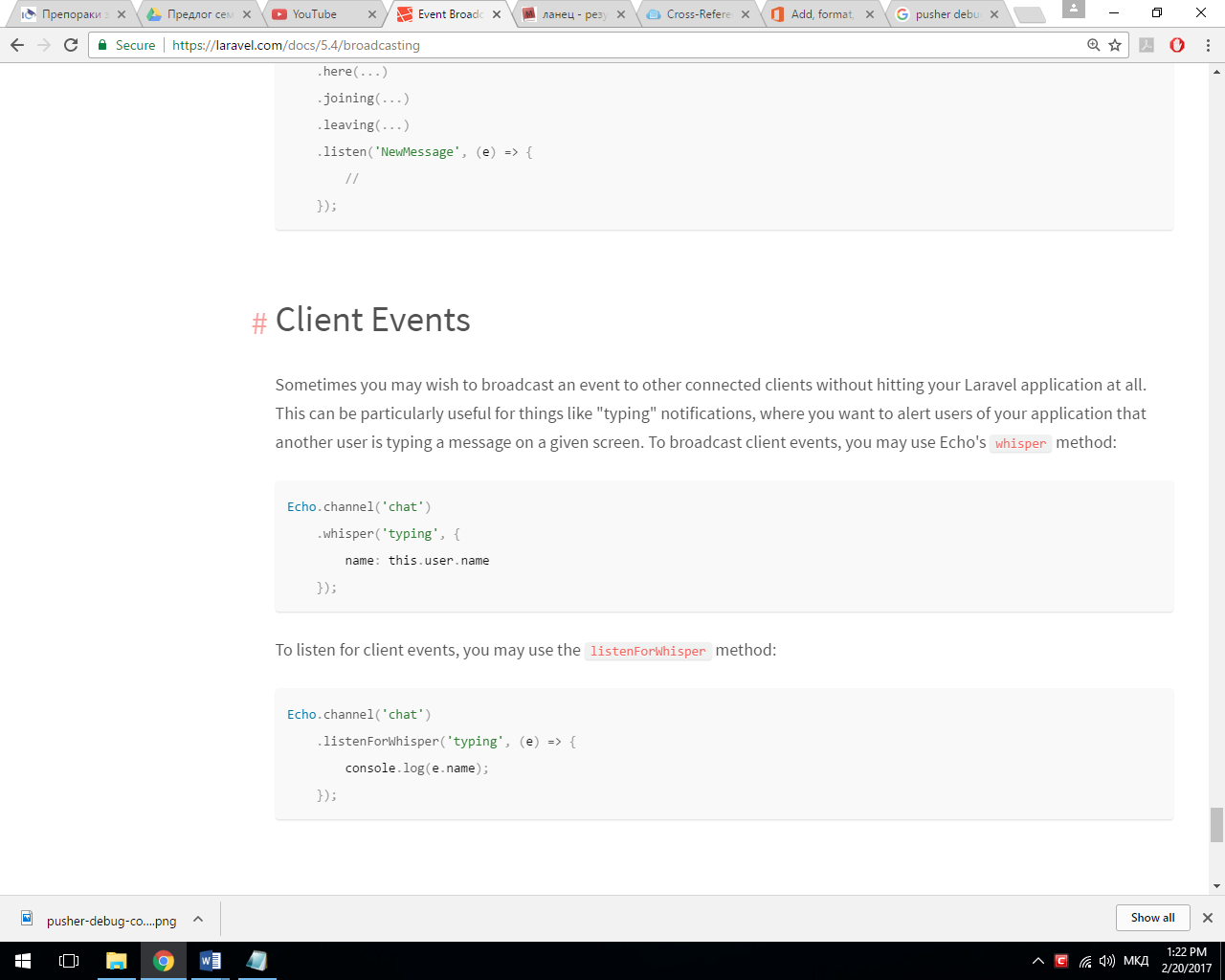
Слика 9 - Претплата на канал за присутност кај Laravel Echo

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. Методот joining() ќе се иврши кога нов корисник ќе се претплати на каналот, додека пак leaving() методата ќе се изврши кога некој од корисниците ќе се одјави од каналот.

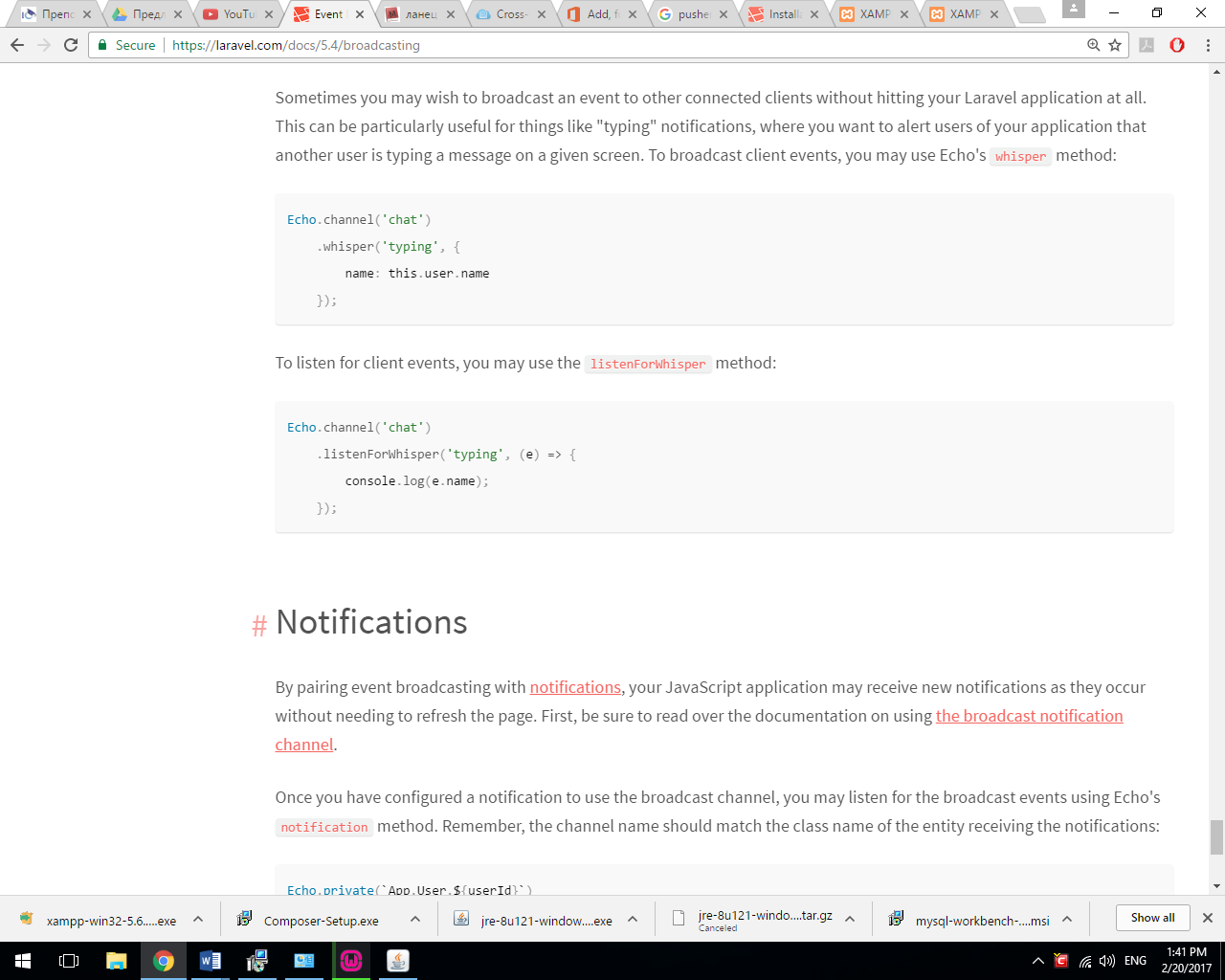
 Исто како и кај приватните и јавни канали, и кај каналите за присутност може да се емитираат настани, со помош на broadcast() и toOthers() соодветно. Методот broadcastOn() треба да врати инстанца од PresenceChannel() класата. На клиентскиот дел, исто така може да слушаме за настани од наш интерес (Слика 10).

Слика 10 - Слушање настани кај канали за присутност

## Клиентски настани

 Понекогаш имаме потреба да емитираме настан до другите претплатени клиенти без притоа да правиме барање до нашата Laravel апликација. 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 11).

Слика 11 - Клиентски настани

 За слушање на клиентски настани, ја користиме listenForWhisper() медодата (Слика 12).

Слика 12 - Слушање клиентски настани

# Методологија

## 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.

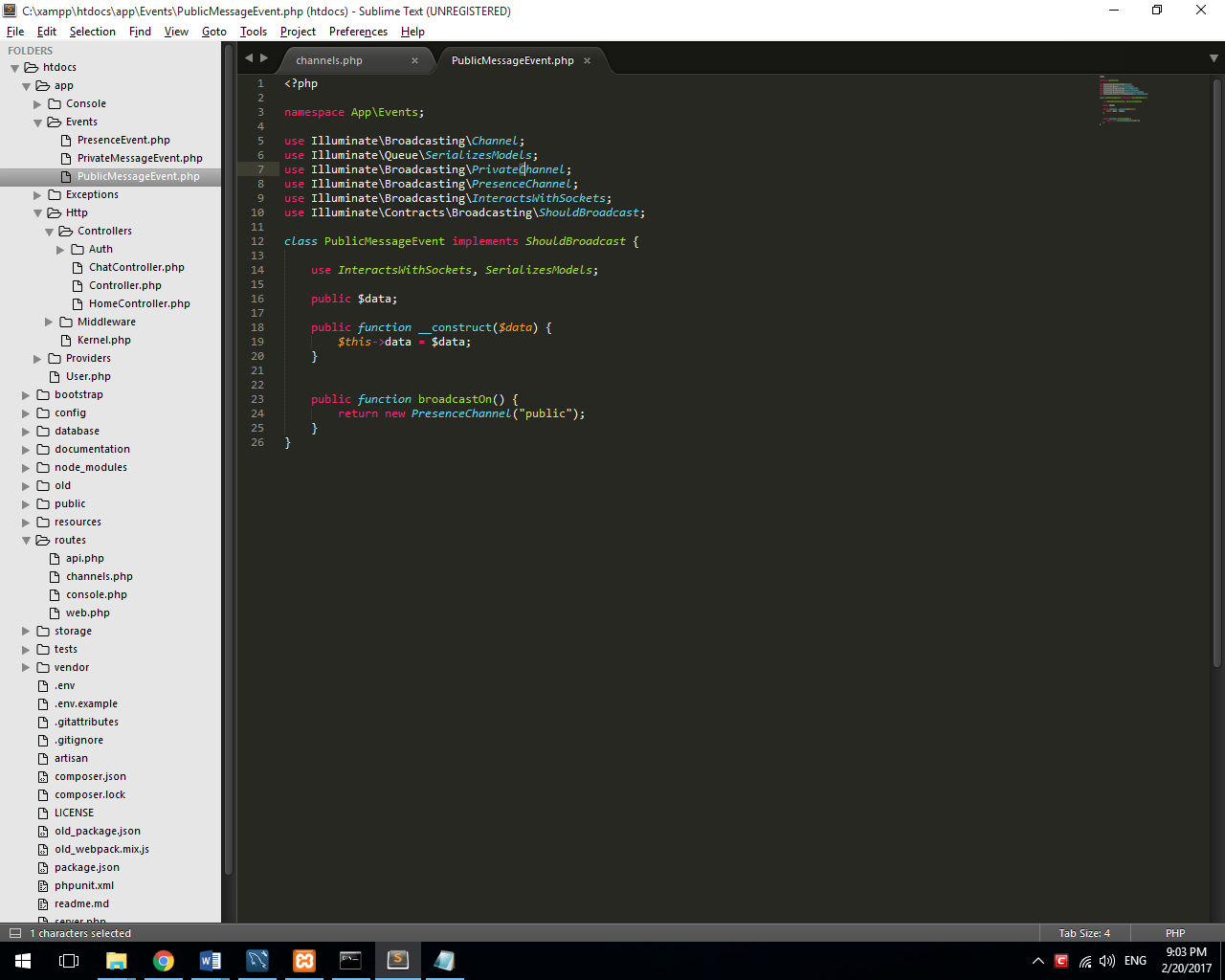
Според спецификациите наведени во документацијата за Laravel Authentication, треба да креираме табела „users“ во која ќе ги чуваме корисниците. За да ја креираме табелата, ќе ги извршиме веќе дефинираните миграции во нашиот проект со помош на наредбата php artisan migrate. Миграциите ќе ги креираат табелите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

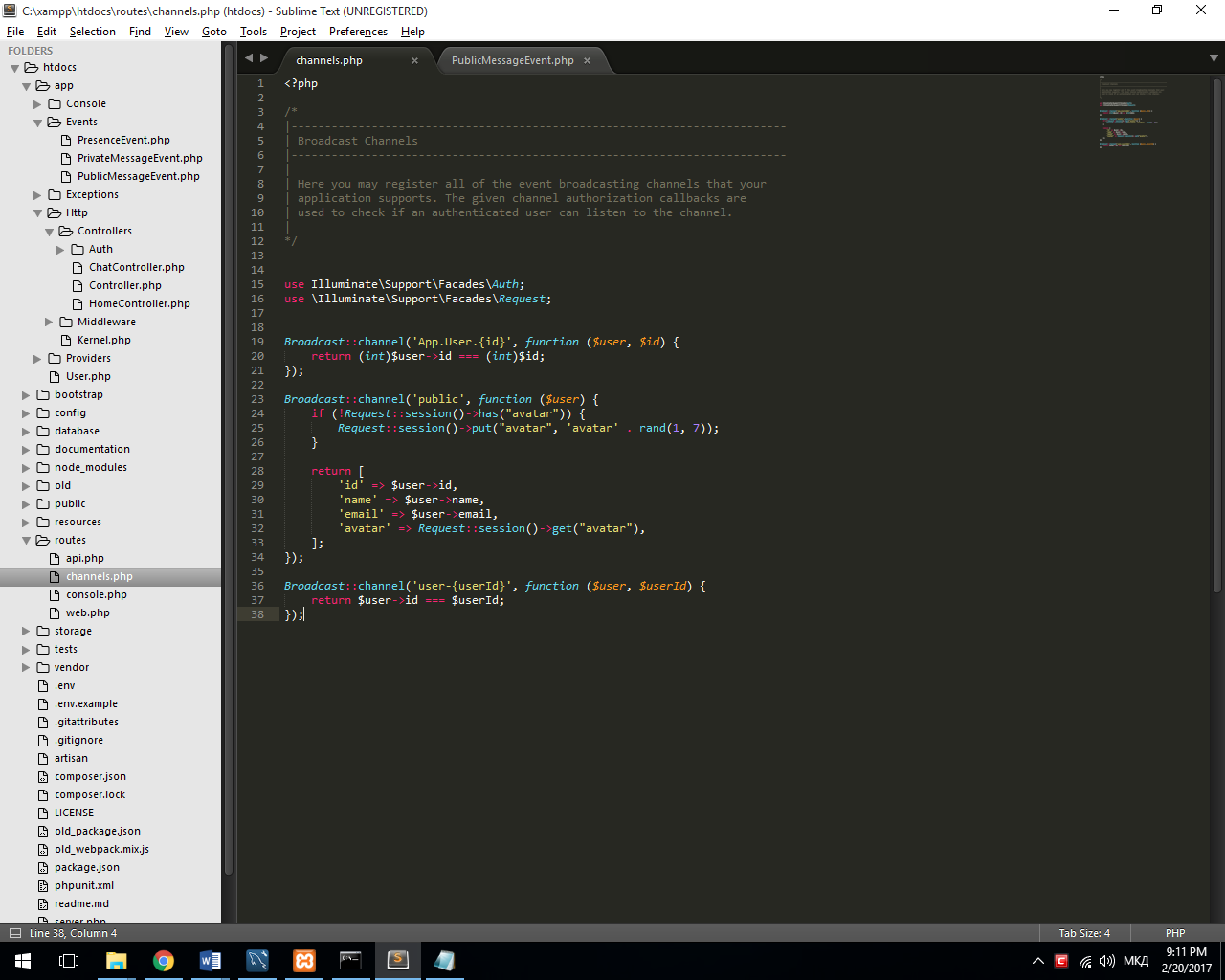
Поради bug во конфигурацијата на моменталната верзија на Laravel Mix, ќе конфигурираме засебна инстанца на Webpack. Конфигурацијата е сместена во датотеката webpack.config.js. Скриптите и зависностите кои што ќе ги користиме се дефинирани во package.json датотеката.

## Дефинирање на настани

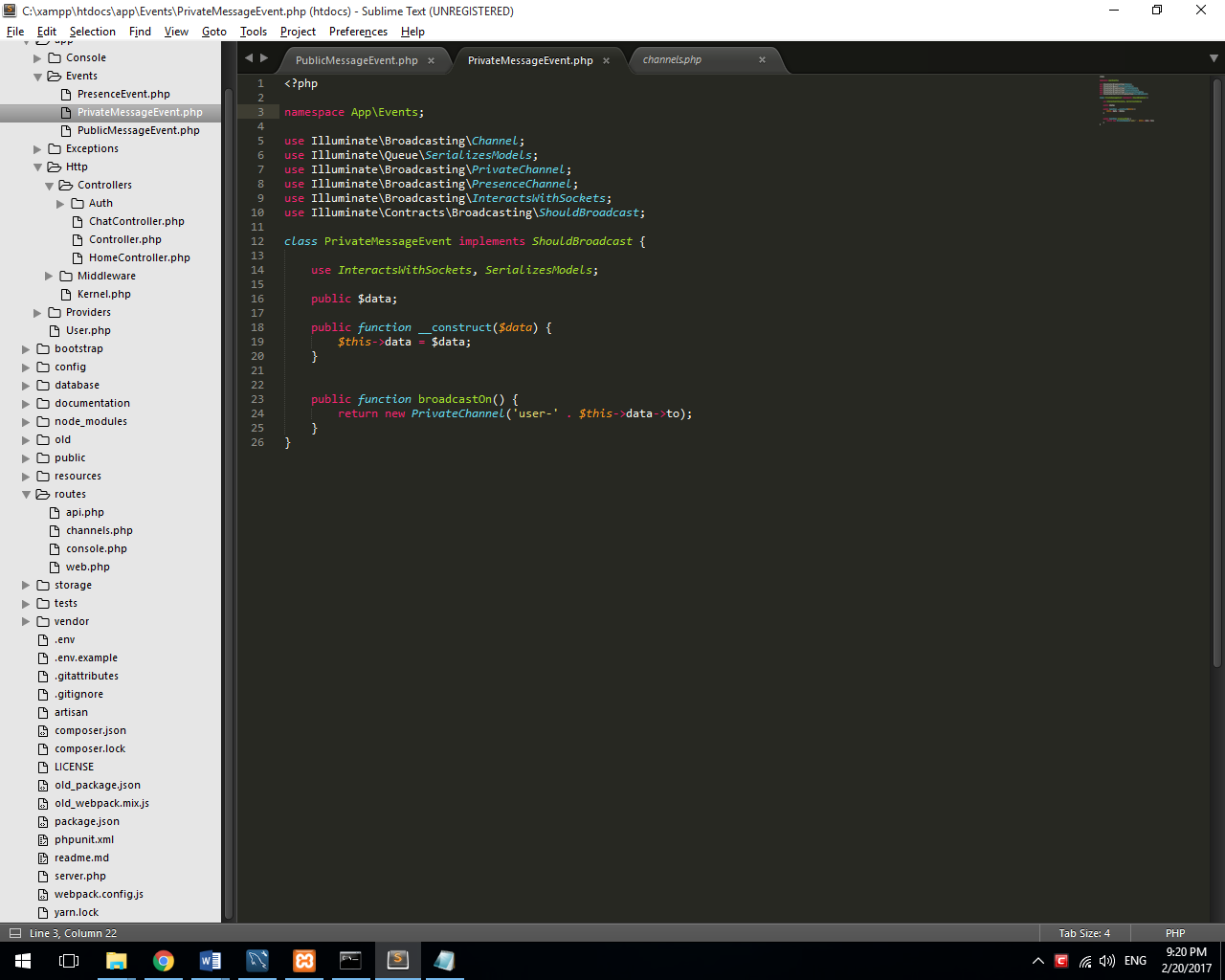
 Нашата апликација има еден приватен канал кој е достапен на сите најавени корисници. На овој канал корисниците ќе можат да се допишуваат меѓу себе и пораките кои што ќе ги праѓаат ќе бидат јавни достапни и видливи за сите корисници. Настанот за праќање на јавни порараки е PublicMessageEvent (Слика 13), а името на каналот е „public“.

Слика 13 - Настан за јавни пораки

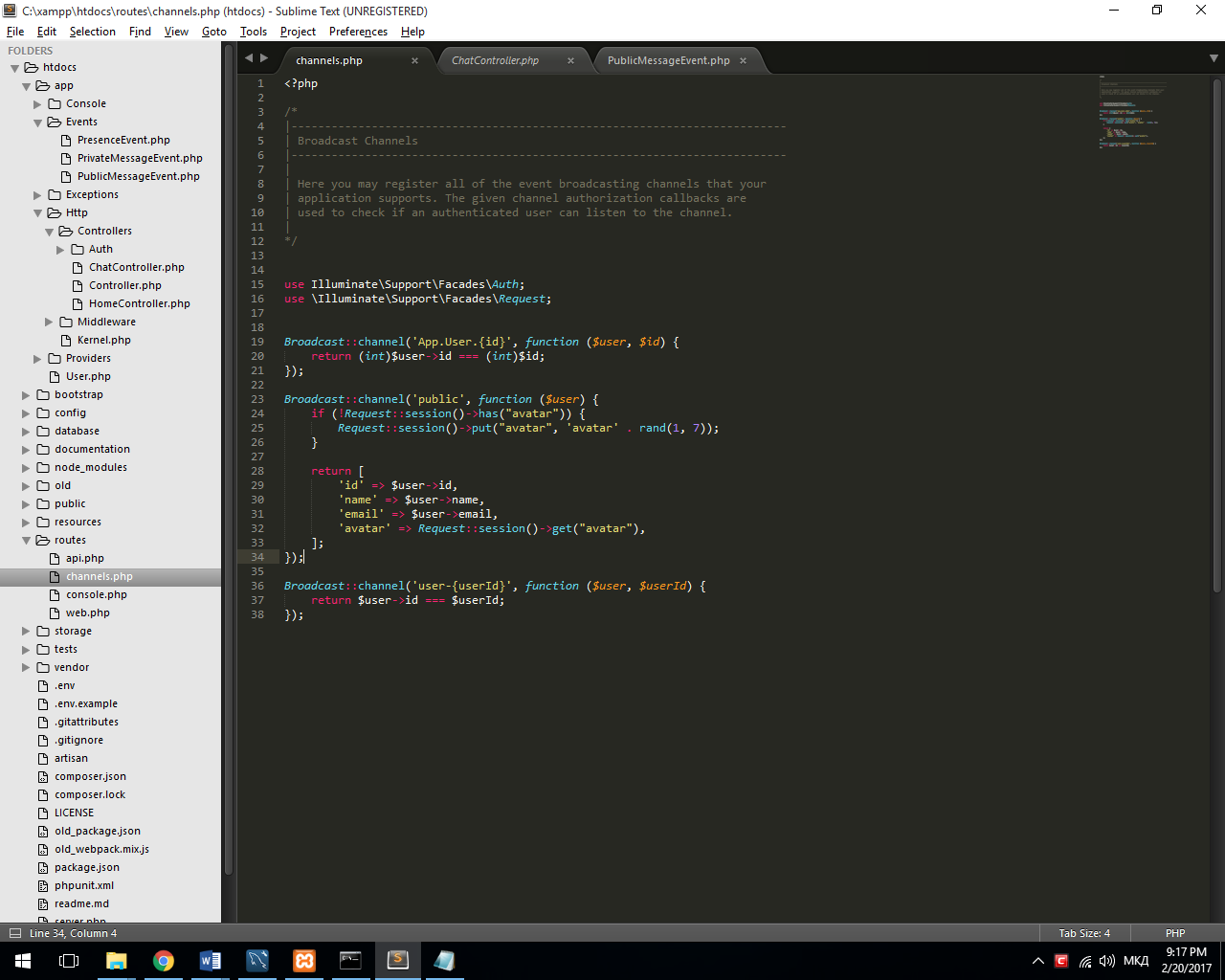
Како што може да забележиме од Слика 13, „public“ каналот е канал за присутност. Како што споменавме на почетокот, секој канал за присутност е всушност и приватен канал. Каналите за присутност ни даваат информација за корисници кои се претплатени на каналот во моментот. Во датотеката channels.php е дефинирана callback функција која враќа низата со податоци за корисникот (Слика 14).



Слика 14 - 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 15). 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.

Слика 15 - Настан за приватни пораки

Во датотеката channels.php е дефинирана callback функција со која дозволува или забранува пристапот на корисникот до соодветниот приватен канал (Слика 16).

Слика 16 - Канал за приватни пораки