WebSocket

- Introduction
- What is a Socket?
- A socket is a software endpoint that establishes communication between two computer systems over a network. It enables data exchange between applications, facilitating real-time communication.
- What is WebSocket?
- WebSocket is a communication protocol that provides full-duplex communication channels over a single, long-lived connection. Unlike traditional HTTP, which follows a request-response model, WebSocket allows for bidirectional communication, enabling both the server and the client to send messages independently.
- Why is WebSocket Needed?

1. Real-time Communication:

1. WebSocket allows for real-time communication between the server and the client, making it ideal for applications that require instant updates, such as chat applications, live notifications, and collaborative editing tools.

2. Reduced Latency:

1. Since WebSocket maintains a persistent connection, there is less overhead in establishing and tearing down connections for each interaction. This results in lower latency compared to traditional polling mechanisms.

3. Efficient Resource Utilization:

1. WebSocket reduces the need for continuous polling requests, which can strain server resources. The persistent connection minimizes the amount of unnecessary data transfer and server load.

- Example:
- Scenario: Real-time Chat Application
- Traditional Approach (Without WebSocket):
- Users must manually refresh the chat page to check for new messages.
- The server needs to handle numerous polling requests, even when there is no new data.
- WebSocket Approach:
- Users receive new messages instantly without refreshing.
- The server only sends data when there is something new, reducing unnecessary requests.

In summary, WebSocket simplifies real-time communication by providing a more efficient and responsive way for applications to exchange data. Its bidirectional nature and reduced latency make it a valuable tool for building modern, interactive web applications.

Socket Server Setup

Steps to set up `beyondcode/laravel-websockets`, `Redis`, `horizon` and `Supervisor`:

1. Install `pusher/pusher-php-server` package:

1. composer require pusher/pusher-php-server "~3.0"

2. Install beyondcode/laravel-websockdets package

1. Add the package to your Laravel project using composer:

1. `composer require beyondcode/laravel-websockets`

2. Publish the configuration file:

1. `php artisan vendor:publish --provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider" --tag="config"`

3. Configure Laravel Websockets

1. Open `config/websockets.php` and configure according to your needs.

2. Add WebSocket server configuration in .env:

```
` WEBSOCKETS_REPLICATION_MODE=redis

PUSHER_APP_ID=dynamicPusherApp

PUSHER_APP_KEY=do77sEhxzW

PUSHER_APP_CLUSTER=mt1

PUSHER_APP_SECRET=80f5dd30-c24e-5432-87ae-d57d0059f109

PUSHER_APP_HOST=192.168.0.76

PUSHER_PUSH_SCHEME=http

PUSHER_APP_PORT=6001

PUSHER_APP_TLS=false

PUSHER_APP_ENCRYPTED=false

`
```

4. Configure your Laravel Broadcasting

1. In `config/broadcasting.php`, set the `host` and `port` under `pusher` options to match your WebSocket server.

1. BROADCAST_DRIVER=pusher # Broadcast driver

5. If you are on HTTPS, then add SSL certificate file for WSS protocol as well in `.env` :

For SSL certification

LARAVEL_WEBSOCKETS_SSL_LOCAL_CERT=/home/ec2-user/dynamicssl/STAR.dynamicpginc.com.crt

LARAVEL_WEBSOCKETS_SSL_LOCAL_PK=/home/ec2-user/dynamicssl/STAR.dynamicpginc.com_key.pem

6. Install and configure Redis server with Laravel

1. Install Redis on your server: `sudo apt-get install redis-server`

2. Install Predis package: `composer require predis/predis`

3. Configure Laravel to use Redis in `.env` file:

```
CACHE_DRIVER=redis
QUEUE_CONNECTION=redis
REDIS_CLIENT=predis

REDIS_HOST=127.0.0.1          # Redis hostname
REDIS_PASSWORD=               # Redis password
REDIS_PORT=6379               # Redis port
```

7. Install and configure Supervisor

1. Install Supervisor on your server: `sudo apt-get install supervisor`

2. Create a new Supervisor configuration file for your Laravel queue worker in `/etc/supervisor/conf.d/` directory. You need to setup two supervisors websocket supervisor and horizon supervisor.

The file might look like this:

```
[program:laravel-worker]
process_name=%(program_name)s_%(process_num)02d
command=php /path-to-your-project/artisan queue:work redis --sleep=3 --tries=3
autostart=true
autorestart=true
user=your-user
numprocs=8
redirect_stderr=true
stdout_logfile=/path-to-your-project/storage/logs/worker.log
```

```

8. Test your setup

1. You can test your setup by visiting `http://your-domain/laravel-websockets` in your browser.

9. Install Laravel Horizon

1. Add Horizon to your Laravel project using composer:

1. `composer require laravel/horizon`

2. Publish the Horizon assets:

1. `php artisan horizon:install`

10. Configure Horizon

1. Open `config/horizon.php` and configure according to your needs. You can define your queue worker configurations here.

11. Update Supervisor Configuration

1. Update your supervisor configuration file to use Horizon's `artisan horizon` command instead of `queue:work`. The command section of your Supervisor configuration file should look like this: No need to change anything

```

command=php /path-to-your-project/artisan horizon

```

12. Start Horizon

1. Start Horizon using the command: `php artisan horizon`

2. Start Using Supervisor: `sudo supervisorctl start  horizon-file-name`

13. Start the WebSocket server

1. Start the WebSocket server using command:

1. `php artisan websockets:serve`

2. Start Using Supervisor: `sudo supervisorctl start  websocket-file-name`

14. Dashboard

1. Horizon provides a dashboard at `/horizon` route. You can change this in the `config/horizon.php` file.

Remember to replace placeholders like `path-to-your-project` and `your-user` with actual values.

Implement Listeners:

1. As per current setup we have used two packages JavaScript packages for that you will require `npm` as package manager.
```

1. "laravel-echo": "^1.13.1"

2. "pusher-js": "^7.3.0"

2. For development purpose you will require

1. "laravel-mix": "^6.0.49"

package to generate listener file from .env configurations.
Note: Use `npm install` command to install required three packages.

3. At final after installing JS packages create code file inside `resources/js`:

```
import Echo from 'laravel-echo';
import Pusher from "pusher-js"
const echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    wsHost: process.env.MIX_PUSHER_APP_HOST,
    wsPort: process.env.MIX_PUSHER_APP_PORT,
    wssPort: process.env.MIX_PUSHER_APP_PORT,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER,
    forceTLS: JSON.parse(process.env.MIX_PUSHER_APP_TLS),
    enabledTransports: ['ws', 'wss'],
    encrypted: JSON.parse(process.env.MIX_PUSHER_APP_ENCRYPTED),
})
const shiftChannel = echo.channel('shift');
shiftChannel.listen('.shift.created', event => shiftCreateHandler(event));
shiftChannel.listen('.shift.updated', event => shiftUpdateHandler(event));
shiftChannel.listen('.shift.deleted', event => shiftDeleteHandler(event));
shiftChannel.listen('.page.refresh', event => pageRefreshHandler(event));

function shiftCreateHandler(data) {
    createShiftBox(data.shiftData);
}
function shiftUpdateHandler(data) {
    updateShiftBox(data.shiftData);
}
```

```
function shiftDeleteHandler(data) {
    deleteShiftBox(data.shiftData);
}
function pageRefreshHandler(data){
    refreshPage(data.message);
}
```

After creating a JS file into resource folder, you need to run following command to generate new executables in public folder: `npx mix`

Cheat sheet

Here's a cheat sheet for maintaining your socket setup:

1. Laravel Websockets

1. Check the status of the WebSocket server:

1. `php artisan websockets:status`

2. Starting the Websockets Server

1. Start the server: `php artisan websockets:serve`

2. Start the server on a specific port: `php artisan websockets:serve --port=6001`

3. Start the server on a specific host: `php artisan websockets:serve --host=your-host`

3. Stopping the Websockets Server

1. Unfortunately, there's no built-in command to stop the server. You'll need to manually kill the process using standard system commands like `kill` or `pkill`.

4. Checking the Status of the Websockets Server

1. Check the status: `php artisan websockets:status`

5. Cleaning Up Old Statistics

1. Clean up old statistics: `php artisan websockets:clean`

6. Debugging

1. If you're having issues, you can start the server in debug mode: `php artisan websockets:serve --debug`

7. Remember, if you make changes to your broadcasting events, you'll need to restart the WebSocket server for those changes to take effect.

2. Redis

1. Monitor Redis: `redis-cli monitor`

2. Clear Redis cache: `redis-cli flushall`
3. Supervisor

1. Reread Supervisor configuration: `sudo supervisorctl reread`
2. Update Supervisor processes: `sudo supervisorctl update`
3. Start all Supervisor processes: `sudo supervisorctl start all`
4. Stop all Supervisor processes: `sudo supervisorctl stop all`
5. Restart all Supervisor processes: `sudo supervisorctl restart all`
6. Check the status of Supervisor processes: `sudo supervisorctl status`

4. Laravel Horizon

1. Start Horizon: `php artisan horizon`
2. List Horizon master supervisors: `php artisan horizon:list`
3. Pause Horizon: `php artisan horizon:pause`
4. Continue Horizon: `php artisan horizon:continue`
5. Terminate Horizon: `php artisan horizon:terminate`
6. Check the status of Horizon: `php artisan horizon:status`

5. General Laravel Maintenance

1. Clear application cache: `php artisan cache:clear`
2. Clear route cache: `php artisan route:clear`
3. Clear config cache: `php artisan config:clear`
4. Clear compiled views: `php artisan view:clear`
5. Run database migrations: `php artisan migrate`
6. Run database seeders: `php artisan db:seed`

⑩ Reference:

https://beyondco.de/docs/laravel-websockets/getting-started/introduction

https://freek.dev/1228-introducing-laravel-websockets-an-easy-to-use-websocket-server-implemented-in-php

https://www.hostinger.in/tutorials/how-to-install-and-setup-redis-on-ubuntu/

https://snyk.io/advisor/npm-package/laravel-echo/example

https://laravel.com/docs/10.x/broadcasting#model-broadcasting-conventions