Server-Sent Events (SSE) in Laravel: un esempio concreto

Le applicazioni web moderne spesso richiedono aggiornamenti in tempo reale per fornire un'esperienza utente dinamica e interattiva. Una delle tecniche per ottenere aggiornamenti in tempo reale dal server al client è l'utilizzo dei Server-Sent Events (SSE). In questo articolo, esploreremo come implementare i Server-Sent Events in un'applicazione Laravel per notificare al frontend il numero totale di ordini in corso.

Cosa sono i Server-Sent Events?

I Server-Sent Events (SSE) sono una tecnologia che permette ai server di inviare aggiornamenti automatici ai client attraverso una connessione HTTP unidirezionale. Diversamente dal WebSocket, che è bidirezionale, SSE è progettato per flussi di dati in tempo reale dal server al client, rendendolo ideale per notifiche o aggiornamenti periodici.

Prerequisiti

Assicurati di avere un'installazione funzionante di Laravel. Puoi creare un nuovo progetto Laravel con il seguente comando:

```
composer create-project --prefer-dist laravel/laravel realtime-orders
```

Configurazione del backend

Creare il model e la migration per gli ordini

Per prima cosa, creiamo il modello Order e la relativa migrazione:

```
php artisan make:model Order -m
```

Apri la migrazione generata in `database/migrations` e aggiungi i campi necessari:

```
public function up()
```

```
{
    Schema::create('orders', function (Blueprint $table) {
        $table->id();
        $table->string('customer_name');
        $table->integer('quantity');
        $table->string('status')->default('progress');
        $table->timestamps();
    });
}
```

Esegui la migrazione:

```
php artisan migrate
```

Creare un controller per gestire gli SSE

Creiamo un controller `SSEOrderController`:

```
php artisan make:controller SSEOrderController
```

Aggiungi un metodo per gestire la connessione SSE:

```
use Illuminate\Http\Request;
use App\Models\Order;
use Symfony\Component\HttpFoundation\StreamedResponse;

class OrderController extends Controller
{
    public function streamOrders(Request $request)
    {
        $response = new StreamedResponse(function () {
            while (true) {
                $orderCount = Order::where('status', 'progress')->count();
                echo "data: " . json_encode(['orderCount' => $orderCount]) .
"\n\n";
                ob_flush();
                flush();
                sleep(5);
            }
        });
```

```
        $response->headers->set('Content-Type', 'text/event-stream');
        $response->headers->set('Cache-Control', 'no-cache');
        $response->headers->set('Connection', 'keep-alive');

        return $response;
    }
}
```

Definire le rotte

Apri il file `routes/api.php` e aggiungi la rotta per il flusso SSE:

```
use App\Http\Controllers\OrderController;

Route::get('/stream-orders', [OrderController::class, 'streamOrders'])-
>name('stream.orders');
```

Configurazione del frontend

Nel frontend, utilizzeremo l'oggetto `EventSource` per ascoltare gli eventi inviati dal server. Possiamo scrivere il seguente codice JavaScript:

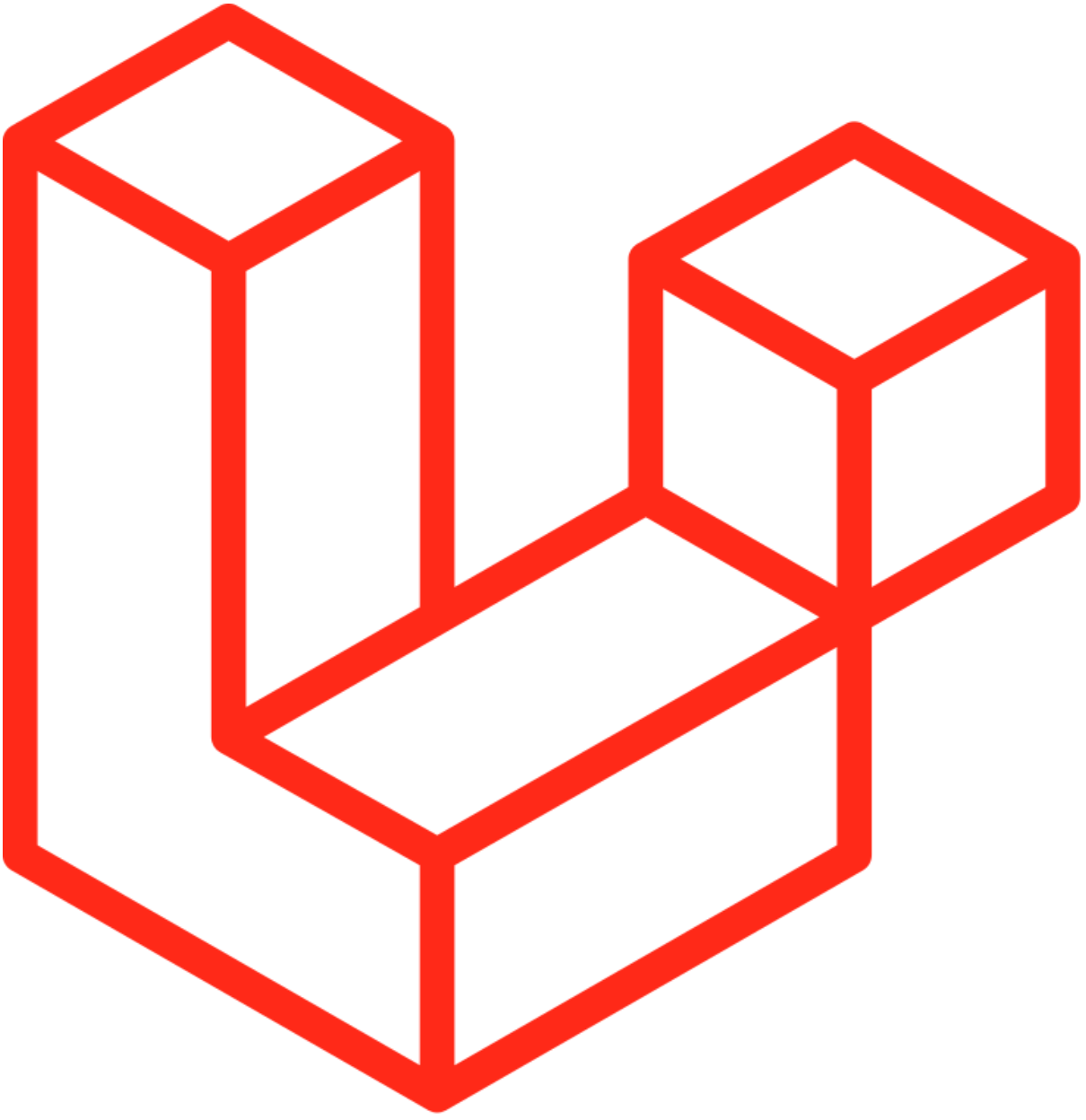
```
const orderCountElement = document.getElementById('order-count');
const eventSource = new EventSource('/api/stream-orders');

eventSource.onmessage = function(event) {
   const data = JSON.parse(event.data);
   orderCountElement.textContent = data.orderCount;
};
```

Conclusione

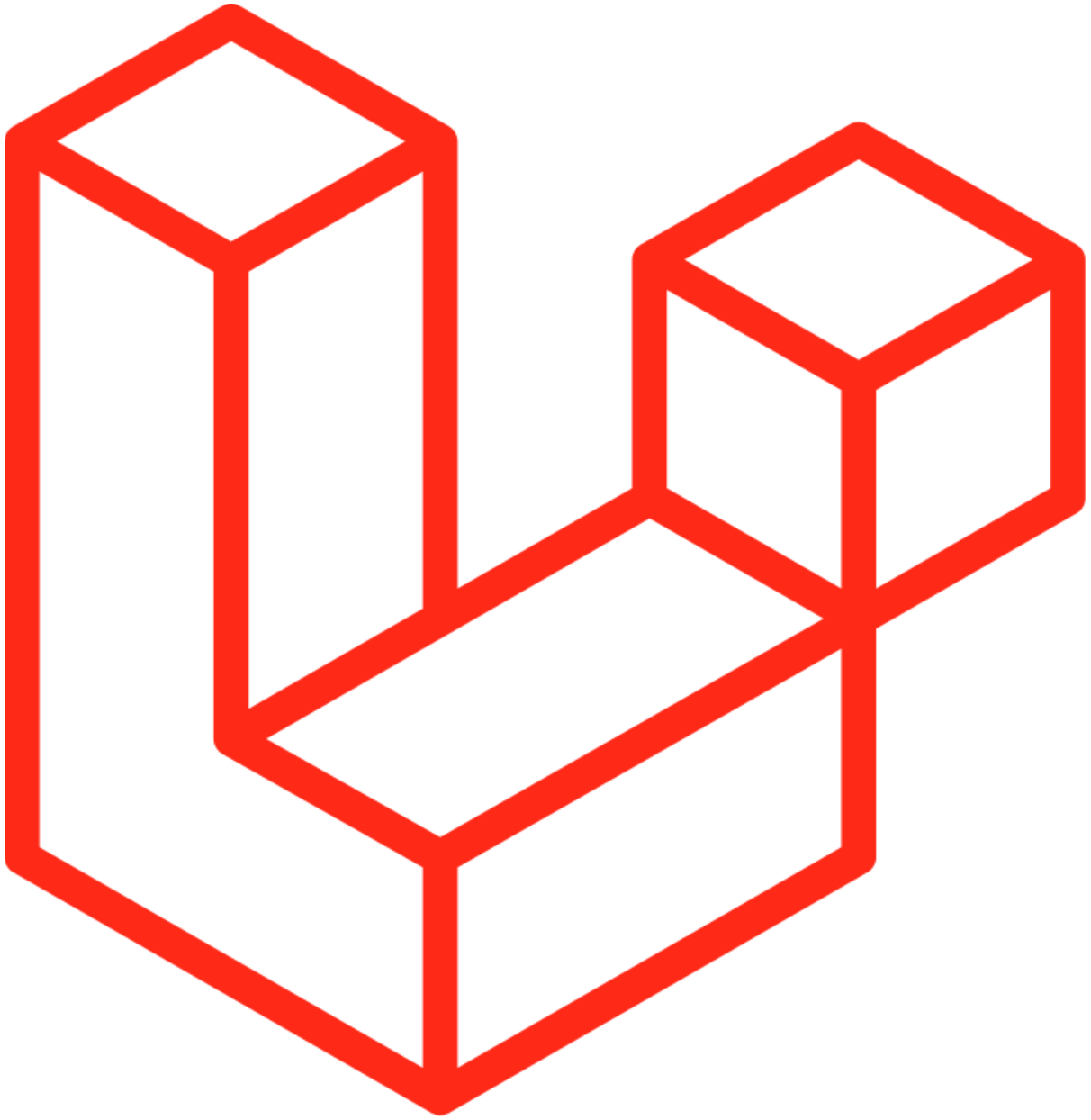
In questo articolo, abbiamo esplorato come utilizzare i Server-Sent Events in un'applicazione Laravel per inviare notifiche in tempo reale al frontend sul numero totale di ordini in corso. Questa implementazione può essere estesa e modificata per altri casi d'uso, come notifiche su aggiornamenti di stock, messaggi in chat in tempo reale, e molto altro. I Server-Sent Events sono una soluzione potente e semplice per migliorare l'interattività e la reattività delle applicazioni web.

Applicazioni Correlate



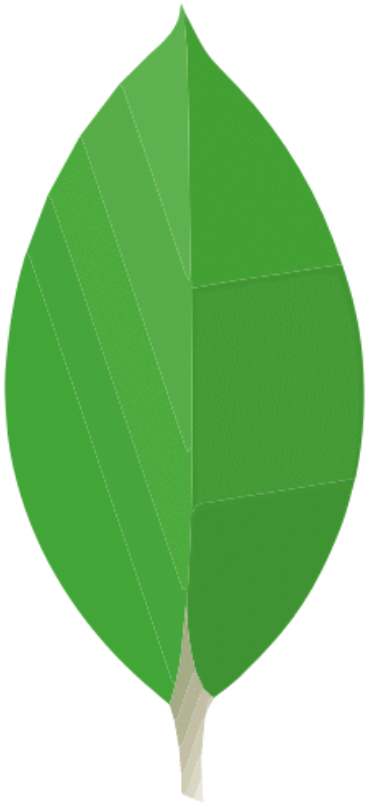
- ### Laravel Placeholder Image

  Applicazione in Laravel per creare immagini segnaposto.
  DockerDocker ComposeComposerPHPLaravelJavaScript



Laravel Shopping Cart

Applicazione che fa parte del progetto Laravel E-commerce e implementa la gestione del carrello in Laravel.
DockerDocker ComposeComposerPHPLaravel



PHP MongoDB App

Applicazione basata su MongoDB con il driver PHP ufficiale.
DockerDocker ComposeComposerPHPMongoDB