Para criar um fluxo de lances em tempo real em uma aplicação **Laravel 9**, utilizando **Socket.IO** para atualizar os lances instantaneamente, siga os passos abaixo:

---

1. Estrutura do Banco de Dados

Crie uma tabela `bids` (lances), relacionando com `users` e `ads` (anúncios):

```
php artisan make:migration create_bids_table
```

Edite a migração:

```
public function up()
{
    Schema::create('bids', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id')->constrained()-
        >onDelete('cascade');
        $table->foreignId('ad_id')->constrained()-
        >onDelete('cascade');
        $table->decimal('amount', 10, 2);
        $table->timestamps();
    });
}
```

Execute:

```
php artisan migrate
```

---

2. Criar o Modelo `Bid.php`

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Bid extends Model
{
    use HasFactory;

    protected $fillable = ['user_id', 'ad_id', 'amount'];

    public function user()
    {
```

```
        return $this->belongsTo(User::class);
    }

    public function ad()
    {
        return $this->belongsTo(Ad::class);
    }
}
```

---

3. Criar Controller `BidController.php`

```
namespace App\Http\Controllers;

use App\Models\Bid;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Events\NewBidEvent;

class BidController extends Controller
{
    public function store(Request $request)
    {
        $request->validate([
            'ad_id' => 'required|exists:ads,id',
            'amount' => 'required|numeric|min:0.01',
        ]);

        $bid = Bid::create([
            'user_id' => Auth::id(),
            'ad_id' => $request->ad_id,
            'amount' => $request->amount,
        ]);

        broadcast(new NewBidEvent($bid))->toOthers(); // Envia
        evento em tempo real

        return response()->json($bid);
    }
}
```

---

4. Criar Evento NewBidEvent.php

```php
namespace App\Events;

use App\Models\Bid;
use Illuminate\Broadcasting\Channel;
use Illuminate\Queue\SerializesModels;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;

class NewBidEvent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $bid;

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

    public function broadcastOn()
    {
        return new Channel('bids');
    }

    public function broadcastWith()
    {
        return [
            'id' => $this->bid->id,
            'user' => $this->bid->user->name,
            'amount' => $this->bid->amount,
            'ad_id' => $this->bid->ad_id,
        ];
    }
}
```

---

5. Configurar Laravel Echo e Socket.IO

Instalar Dependências

```
composer require predis/predis
npm install --save laravel-echo socket.io-client
```

Configurar `broadcasting.php`

No arquivo `config/broadcasting.php`, configure **Pusher** como driver padrão:

```
'default' => env('BROADCAST_DRIVER', 'pusher'),
```

No `.env`, adicione:

```
BROADCAST_DRIVER=pusher
PUSHER_APP_ID=your_app_id
PUSHER_APP_KEY=your_key
PUSHER_APP_SECRET=your_secret
PUSHER_HOST=127.0.0.1
PUSHER_PORT=6001
PUSHER_SCHEME=http
```

Se você deseja usar **broadcasting** em **Laravel 9** com opções **gratuitas**, você pode optar por **Redis** ou **Socket.IO** em vez de Pusher. Aqui estão algumas alternativas gratuitas:

---

1. Usando Redis (Grátis)

O **Redis** é uma ótima alternativa para broadcast em tempo real sem custos adicionais.

Instalar Redis no Laravel

```
composer require predis/predis
```

Configurar `.env`

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

Configurar `config/broadcasting.php`

```
'default' => env('BROADCAST_DRIVER', 'redis'),
```

Rodar o servidor Redis (Se necessário)

Se estiver localmente, inicie o Redis:

```
redis-server
```

Rodar o Worker do Laravel

```
php artisan queue:work
```

Agora você pode fazer broadcast de eventos **sem precisar de serviços pagos** como Pusher.

---

2. Usando Socket.IO (WebSockets Customizados - Grátis)

Se você não quiser usar Redis, pode configurar um servidor **WebSockets** manualmente com **Socket.IO**.

Instalar Dependências

```
npm install socket.io socket.io-client
```

Criar Servidor WebSocket (`server.js`)

```
const io = require("socket.io")(6001, {
    cors: {
        origin: "*",
    },
});

io.on("connection", (socket) => {
    console.log("Usuário conectado:", socket.id);

    socket.on("newBid", (data) => {
        io.emit("updateBids", data);
    });

    socket.on("disconnect", () => {
        console.log("Usuário desconectado:", socket.id);
    });
```

```
});

console.log("Servidor WebSocket rodando na porta 6001...");
```

Alterar `.env` para WebSockets

```
BROADCAST_DRIVER=log
```

Integrar com Laravel Echo no Frontend

```
import io from "socket.io-client";

const socket = io("http://127.0.0.1:6001");

socket.on("updateBids", (data) => {
    console.log("Novo lance recebido:", data);
});
```

Agora, ao enviar um lance, o evento será transmitido via **Socket.IO**, sem necessidade de serviços pagos.

Conclusão

Melhor opção gratuita para alta performance? Redis

Melhor opção gratuita para WebSockets sem Redis? Socket.IO

Se sua aplicação precisar de escalabilidade, **Redis + Laravel Echo** é a melhor solução **grátis**. Se quiser simplicidade sem Redis, **Socket.IO** é uma boa escolha.

Criar o Servidor WebSocket (`server.js`)

Crie um arquivo `server.js` na raiz do projeto:

```
const io = require("socket.io")(6001, {
    cors: {
        origin: "*",
    },
});

console.log("Servidor WebSocket rodando na porta 6001...");

io.on("connection", (socket) => {
```

```
    console.log("Novo usuário conectado");

    socket.on("disconnect", () => {
        console.log("Usuário desconectado");
    });
});
```

Inicie o WebSocket:

```
node server.js
```

---

6. Criar Rotas

Edite `routes/api.php`:

```
use App\Http\Controllers\BidController;

Route::middleware('auth:sanctum')->post('/bids',
        [BidController::class, 'store']);
```

---

7. Criar o Frontend (Blade com jQuery)

Com jQuery (Blade)

No seu Blade (`bids.blade.php`):

```
<input type="number" id="amount" placeholder="Valor do lance">
<button onclick="placeBid()">Enviar Lance</button>

<ul id="bids-list"></ul>

<script src="https://cdn.socket.io/4.0.1/socket.io.min.js"></
        script>
<script>
    const socket = io("http://127.0.0.1:6001");

    socket.on("bids", function (data) {
        document.getElementById("bids-list").innerHTML +=
            "<li>" + data.user + " fez um lance de R$ " +
        data.amount + "</li>";
    });

    function placeBid() {
        let amount = document.getElementById("amount").value;
```

```
        fetch("/api/bids", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
                Authorization: "Bearer SEU_TOKEN_AQUI",
            },
            body: JSON.stringify({ ad_id: 1, amount: amount }),
        });
    }
</script>
```

---

8. Executar a Aplicação

```
php artisan serve
```

Em outro terminal:

```
node server.js
```

Agora, ao enviar um lance, todos os usuários conectados verão a atualização em tempo real!