Chatwoot Agent Dashboard — Structure Laravel MVP

Architecture Projet

```
chatwoot-agent/
  app/
      Http/
          Controllers/
              Auth/
                  LoginController.php          # Authentification agents
              ConversationController.php        # CRUD conversations
              MessageController.php             # Envoi/réception messages
              DashboardController.php           # Stats & reporting
              AgentController.php               # Gestion agents
              ContactController.php             # Gestion contacts
              Webhook/
                  TwilioWebhookController.php   # Handoff Twilio → Chatwoot
                  ChatwootWebhookController.php # Events Chatwoot → Twilio
          Middleware/
              ValidateTwilioSignature.php       # Sécurité webhook Twilio
          Requests/
              SendMessageRequest.php
              AssignConversationRequest.php
      Services/
          Chatwoot/
              ChatwootClient.php                # Client HTTP API Chatwoot
              ConversationService.php           # Logique conversations
              MessageService.php                # Logique messages
              ContactService.php                # Logique contacts
              AgentService.php                  # Logique agents
              ReportService.php                 # Logique reporting
          Twilio/
              TwilioService.php                 # Envoi messages WhatsApp
      DTOs/
          ConversationDTO.php                   # Data Transfer Object
          MessageDTO.php
          ContactDTO.php
      Enums/
          ConversationStatus.php                # open, pending, resolved, snoozed
          MessageType.php                       # incoming (0), outgoing (1), activity (2)
          AssigneeType.php                      # me, unassigned, all, assigned
  config/
      chatwoot.php                              # Configuration Chatwoot
  routes/
```

```
    web.php                                          # Routes pages agents
    api.php                                          # Routes webhooks
  resources/
    views/
        layouts/
            app.blade.php                           # Layout principal
        auth/
            login.blade.php
        conversations/
            index.blade.php                         # Liste conversations
            show.blade.php                          # Fil de discussion
        dashboard/
            index.blade.php                         # Stats & métriques
        agents/
            index.blade.php                          # Gestion agents
  database/
    migrations/
        create_agent_tokens_table.php               # Tokens Chatwoot des agents
```

---

1. Configuration — `config/chatwoot.php`

```php
<?php

return [
    /*
    |--------------------------------------------------------------------------
    | Chatwoot API Configuration
    |--------------------------------------------------------------------------
    |
    | URL de base de votre instance Chatwoot self-hosted
    | Account ID du compte principal
    | Token d'accès API (user_access_token ou platform_app_token)
    |
    */

    'base_url'    => env('CHATWOOT_BASE_URL', 'https://support.yeswechange.com'),
    'account_id'  => env('CHATWOOT_ACCOUNT_ID'),
    'api_token'   => env('CHATWOOT_API_TOKEN'),

    // Platform API (pour gestion multi-tenant future)
    'platform_token' => env('CHATWOOT_PLATFORM_TOKEN'),

    // Inbox WhatsApp (créé dans Chatwoot, type API Channel)
    'whatsapp_inbox_id' => env('CHATWOOT_WHATSAPP_INBOX_ID'),
```

```
    // Polling interval en millisecondes (côté frontend)
    'polling_interval' => env('CHATWOOT_POLLING_INTERVAL', 4000),

    // Webhook secret pour valider les requêtes Chatwoot
    'webhook_secret' => env('CHATWOOT_WEBHOOK_SECRET'),
];
```

.env correspondant :

```
CHATWOOT_BASE_URL=https://support.yeswechange.com
CHATWOOT_ACCOUNT_ID=1
CHATWOOT_API_TOKEN=votre_user_access_token
CHATWOOT_PLATFORM_TOKEN=votre_platform_app_token
CHATWOOT_WHATSAPP_INBOX_ID=3
CHATWOOT_POLLING_INTERVAL=4000
CHATWOOT_WEBHOOK_SECRET=secret_webhook
```

---

2. Client HTTP API — `ChatwootClient.php`

C'est la couche de base qui fait tous les appels HTTP vers Chatwoot.

```php
<?php

namespace App\Services\Chatwoot;

use Illuminate\Support\Facades\Http;
use Illuminate\Http\Client\PendingRequest;
use Illuminate\Http\Client\Response;

class ChatwootClient
{
    private string $baseUrl;
    private int $accountId;
    private string $token;

    public function __construct()
    {
        $this->baseUrl   = config('chatwoot.base_url');
        $this->accountId = config('chatwoot.account_id');
        $this->token     = config('chatwoot.api_token');
    }

    /**
     * Client HTTP pré-configuré avec auth et base URL
     */
```

```php
private function client(): PendingRequest
{
    return Http::baseUrl($this->baseUrl)
        ->withHeaders([
            'api_access_token' => $this->token,
            'Content-Type'     => 'application/json',
        ])
        ->timeout(15);
}

/**
 * Préfixe automatique du account_id dans les URLs
 */
private function accountUrl(string $path): string
{
    return "/api/v1/accounts/{$this->accountId}/{$path}";
}

//
// CONVERSATIONS
//

/**
 * Lister les conversations avec filtres
 *
 * @param string $status     open|resolved|pending|snoozed|all
 * @param string $assigneeType me|unassigned|all|assigned
 * @param int    $page
 * @param int|null $inboxId
 */
public function listConversations(
    string $status = 'open',
    string $assigneeType = 'all',
    int $page = 1,
    ?int $inboxId = null
): array {
    $query = [
        'status'        => $status,
        'assignee_type' => $assigneeType,
        'page'          => $page,
    ];

    if ($inboxId) {
        $query['inbox_id'] = $inboxId;
    }
```

```
        return $this->client()
            ->get($this->accountUrl('conversations'), $query)
            ->json();
    }

    /**
     * Détails d'une conversation
     */
    public function getConversation(int $conversationId): array
    {
        return $this->client()
            ->get($this->accountUrl("conversations/{$conversationId}"))
            ->json();
    }

    /**
     * Créer une nouvelle conversation (handoff depuis Twilio)
     *
     * @param string $sourceId   Identifiant unique du contact (ex: whatsapp:+225XXXXXXXX)
     * @param int    $inboxId    ID de l'inbox API Channel
     * @param int    $contactId  ID du contact Chatwoot
     * @param string|null $initialMessage  Premier message du client
     */
    public function createConversation(
        string $sourceId,
        int $inboxId,
        int $contactId,
        ?string $initialMessage = null
    ): array {
        $payload = [
            'source_id'  => $sourceId,
            'inbox_id'   => $inboxId,
            'contact_id' => $contactId,
            'status'     => 'open',
        ];

        if ($initialMessage) {
            $payload['message'] = [
                'content' => $initialMessage,
            ];
        }

        return $this->client()
            ->post($this->accountUrl('conversations'), $payload)
            ->json();
    }
```

```php
/**
 * Changer le statut d'une conversation
 *
 * @param string $status open|resolved|pending|snoozed
 */
public function toggleConversationStatus(int $conversationId, string $status): array
{
    return $this->client()
        ->post($this->accountUrl("conversations/{$conversationId}/toggle_status"), [
            'status' => $status,
        ])
        ->json();
}

/**
 * Assigner une conversation à un agent
 */
public function assignConversation(int $conversationId, int $agentId): array
{
    return $this->client()
        ->post($this->accountUrl("conversations/{$conversationId}/assignments"), [
            'assignee_id' => $agentId,
        ])
        ->json();
}

/**
 * Filtrer les conversations (recherche avancée)
 */
public function filterConversations(array $filters, int $page = 1): array
{
    return $this->client()
        ->post($this->accountUrl('conversations/filter'), [
            'payload' => $filters,
            'page'    => $page,
        ])
        ->json();
}

/**
 * Compteurs de conversations
 */
public function getConversationCounts(string $status = 'open', string $assigneeType = 'a
{
    return $this->client()
```

```
        ->get($this->accountUrl('conversations/meta'), [
            'status'        => $status,
            'assignee_type' => $assigneeType,
        ])
        ->json();
}

//
// MESSAGES
//

/**
 * Récupérer les messages d'une conversation
 */
public function getMessages(int $conversationId): array
{
    return $this->client()
        ->get($this->accountUrl("conversations/{$conversationId}/messages"))
        ->json();
}

/**
 * Envoyer un message dans une conversation
 *
 * @param int    $conversationId
 * @param string $content       Texte du message
 * @param bool   $isPrivate     Note interne (visible agents uniquement)
 * @param string $messageType   outgoing|incoming
 */
public function sendMessage(
    int $conversationId,
    string $content,
    bool $isPrivate = false,
    string $messageType = 'outgoing'
): array {
    return $this->client()
        ->post($this->accountUrl("conversations/{$conversationId}/messages"), [
            'content'      => $content,
            'message_type' => $messageType,
            'private'      => $isPrivate,
        ])
        ->json();
}

//
// CONTACTS
```

```php
//

/**
 * Rechercher un contact par numéro de téléphone
 */
public function searchContacts(string $query): array
{
    return $this->client()
        ->get($this->accountUrl('contacts/search'), [
            'q' => $query,
        ])
        ->json();
}

/**
 * Créer un contact
 */
public function createContact(
    string $name,
    string $phoneNumber,
    ?string $email = null,
    int $inboxId = null
): array {
    $payload = [
        'name'         => $name,
        'phone_number' => $phoneNumber,
        'inbox_id'     => $inboxId ?? config('chatwoot.whatsapp_inbox_id'),
    ];

    if ($email) {
        $payload['email'] = $email;
    }

    return $this->client()
        ->post($this->accountUrl('contacts'), $payload)
        ->json();
}

/**
 * Détails d'un contact
 */
public function getContact(int $contactId): array
{
    return $this->client()
        ->get($this->accountUrl("contacts/{$contactId}"))
        ->json();
```

```php
}

//
// AGENTS
//

/**
 * Lister les agents du compte
 */
public function listAgents(): array
{
    return $this->client()
        ->get($this->accountUrl('agents'))
        ->json();
}

//
// REPORTS & STATS
//

/**
 * Rapport par agent
 *
 * @param string $metric  conversations_count|avg_first_response_time|avg_resolution_ti
 * @param string $since   Date début (ISO 8601)
 * @param string $until   Date fin (ISO 8601)
 */
public function getAgentReport(string $metric, string $since, string $until): array
{
    return $this->client()
        ->get($this->accountUrl('reports/agents'), [
            'metric' => $metric,
            'since'  => $since,
            'until'  => $until,
        ])
        ->json();
}

/**
 * Rapport global du compte
 */
public function getAccountReport(string $metric, string $since, string $until): array
{
    return $this->client()
        ->get($this->accountUrl('reports/account'), [
            'metric' => $metric,
```

```
                'since'  => $since,
                'until'  => $until,
            ])
            ->json();
    }

    /**
     * Événements de reporting d'une conversation
     * (first_response_time, resolution_time, etc.)
     */
    public function getConversationReportingEvents(int $conversationId): array
    {
        return $this->client()
            ->get($this->accountUrl("conversations/{$conversationId}/reporting_events"))
            ->json();
    }
}
```

---

3. Services Métier

ConversationService.php

```
<?php

namespace App\Services\Chatwoot;

use App\DTOs\ConversationDTO;

class ConversationService
{
    public function __construct(
        private ChatwootClient $client
    ) {}

    /**
     * Récupérer les conversations filtrées pour l'affichage
     */
    public function getConversations(
        string $status = 'open',
        string $assigneeType = 'all',
        int $page = 1
    ): array {
        $response = $this->client->listConversations($status, $assigneeType, $page);

        return [
```

```
            'conversations' => collect($response['data']['payload'] ?? [])
                ->map(fn($conv) => ConversationDTO::fromArray($conv))
                ->all(),
            'meta' => $response['data']['meta'] ?? [],
        ];
    }

    /**
     * Récupérer une conversation avec ses messages
     */
    public function getConversationWithMessages(int $conversationId): array
    {
        $conversation = $this->client->getConversation($conversationId);
        $messages     = $this->client->getMessages($conversationId);

        return [
            'conversation' => ConversationDTO::fromArray($conversation),
            'messages'     => $messages['payload'] ?? [],
            'contact'      => $messages['meta']['contact']['payload'][0] ?? null,
            'assignee'     => $messages['meta']['assignee'] ?? null,
        ];
    }

    /**
     * Polling : récupérer uniquement les nouveaux messages
     * Appelé par le frontend toutes les 3-5 secondes
     */
    public function getNewMessages(int $conversationId, int $lastMessageId): array
    {
        $messages = $this->client->getMessages($conversationId);
        $allMessages = $messages['payload'] ?? [];

        // Filtrer uniquement les messages plus récents que le dernier connu
        return collect($allMessages)
            ->filter(fn($msg) => $msg['id'] > $lastMessageId)
            ->values()
            ->all();
    }

    /**
     * Résoudre une conversation
     */
    public function resolve(int $conversationId): array
    {
        return $this->client->toggleConversationStatus($conversationId, 'resolved');
    }
```

```
    /**
     * Réouvrir une conversation
     */
    public function reopen(int $conversationId): array
    {
        return $this->client->toggleConversationStatus($conversationId, 'open');
    }

    /**
     * Assigner une conversation
     */
    public function assign(int $conversationId, int $agentId): array
    {
        return $this->client->assignConversation($conversationId, $agentId);
    }

    /**
     * Compteurs pour le sidebar
     */
    public function getCounts(): array
    {
        return $this->client->getConversationCounts();
    }
}
```

MessageService.php

```
<?php

namespace App\Services\Chatwoot;

class MessageService
{
    public function __construct(
        private ChatwootClient $client
    ) {}

    /**
     * Envoyer un message au client (visible sur WhatsApp)
     */
    public function sendToCustomer(int $conversationId, string $content): array
    {
        return $this->client->sendMessage(
            conversationId: $conversationId,
            content: $content,
```

```
            isPrivate: false,
            messageType: 'outgoing'
        );
    }

    /**
     * Envoyer une note interne (visible agents uniquement)
     */
    public function sendPrivateNote(int $conversationId, string $content): array
    {
        return $this->client->sendMessage(
            conversationId: $conversationId,
            content: $content,
            isPrivate: true
        );
    }
}
```

ReportService.php

```
<?php

namespace App\Services\Chatwoot;

use Carbon\Carbon;

class ReportService
{
    public function __construct(
        private ChatwootClient $client
    ) {}

    /**
     * Dashboard stats : résumé de la journée/semaine
     */
    public function getDashboardStats(string $period = 'today'): array
    {
        [$since, $until] = $this->resolvePeriod($period);

        $conversationsCount = $this->client->getAccountReport(
            'conversations_count', $since, $until
        );

        $avgFirstResponse = $this->client->getAccountReport(
            'avg_first_response_time', $since, $until
        );
```

```
        $avgResolution = $this->client->getAccountReport(
            'avg_resolution_time', $since, $until
        );

        $counts = $this->client->getConversationCounts();

        return [
            'conversations_count'    => $conversationsCount,
            'avg_first_response'     => $avgFirstResponse,
            'avg_resolution_time'    => $avgResolution,
            'open_count'             => $counts['data']['meta']['open'] ?? 0,
            'pending_count'          => $counts['data']['meta']['pending'] ?? 0,
            'unassigned_count'       => $counts['data']['meta']['unassigned'] ?? 0,
        ];
    }

    /**
     * Performance par agent
     */
    public function getAgentPerformance(string $period = 'week'): array
    {
        [$since, $until] = $this->resolvePeriod($period);

        return $this->client->getAgentReport(
            'conversations_count', $since, $until
        );
    }

    /**
     * Convertir un label de période en dates ISO
     */
    private function resolvePeriod(string $period): array
    {
        return match ($period) {
            'today' => [
                Carbon::today()->toIso8601String(),
                Carbon::now()->toIso8601String(),
            ],
            'week' => [
                Carbon::now()->startOfWeek()->toIso8601String(),
                Carbon::now()->toIso8601String(),
            ],
            'month' => [
                Carbon::now()->startOfMonth()->toIso8601String(),
                Carbon::now()->toIso8601String(),
```

```
            ],
            default => [
                Carbon::today()->toIso8601String(),
                Carbon::now()->toIso8601String(),
            ],
        };
    }
}
```

---

4. DTO (Data Transfer Object)

ConversationDTO.php

```php
<?php

namespace App\DTOs;

class ConversationDTO
{
    public function __construct(
        public int $id,
        public string $status,
        public ?string $contactName,
        public ?string $contactPhone,
        public ?string $contactThumbnail,
        public ?string $lastMessage,
        public ?string $lastMessageAt,
        public int $unreadCount,
        public ?string $assigneeName,
        public ?int $assigneeId,
        public ?string $inboxName,
        public ?string $priority,
        public array $labels,
        public string $createdAt,
    ) {}

    public static function fromArray(array $data): self
    {
        $sender   = $data['meta']['sender'] ?? [];
        $assignee = $data['meta']['assignee'] ?? [];
        $lastMsg  = $data['last_non_activity_message'] ?? $data['messages'][0] ?? [];

        return new self(
            id: $data['id'],
            status: $data['status'] ?? 'open',
```

```
            contactName: $sender['name'] ?? 'Inconnu',
            contactPhone: $sender['phone_number'] ?? null,
            contactThumbnail: $sender['thumbnail'] ?? null,
            lastMessage: $lastMsg['content'] ?? null,
            lastMessageAt: isset($lastMsg['created_at'])
                ? date('Y-m-d H:i:s', $lastMsg['created_at'])
                : null,
            unreadCount: $data['unread_count'] ?? 0,
            assigneeName: $assignee['name'] ?? null,
            assigneeId: $assignee['id'] ?? null,
            inboxName: $data['meta']['channel'] ?? null,
            priority: $data['priority'] ?? null,
            labels: $data['labels'] ?? [],
            createdAt: isset($data['created_at'])
                ? date('Y-m-d H:i:s', $data['created_at'])
                : now()->toDateTimeString(),
        );
    }
}
```

---

5. Controllers

ConversationController.php

```
<?php

namespace App\Http\Controllers;

use App\Services\Chatwoot\ConversationService;
use App\Services\Chatwoot\MessageService;
use Illuminate\Http\Request;
use Illuminate\Http\JsonResponse;

class ConversationController extends Controller
{
    public function __construct(
        private ConversationService $conversationService,
        private MessageService $messageService,
    ) {}

    /**
     * Page liste des conversations
     */
    public function index(Request $request)
    {
```

```
        $status       = $request->get('status', 'open');
        $assigneeType = $request->get('assignee_type', 'all');
        $page         = $request->get('page', 1);

        $data = $this->conversationService->getConversations(
            $status, $assigneeType, $page
        );

        return view('conversations.index', [
            'conversations' => $data['conversations'],
            'meta'          => $data['meta'],
            'currentStatus' => $status,
            'currentAssignee' => $assigneeType,
        ]);
    }

    /**
     * Page fil de discussion
     */
    public function show(int $conversationId)
    {
        $data = $this->conversationService->getConversationWithMessages($conversationId);

        return view('conversations.show', [
            'conversation' => $data['conversation'],
            'messages'     => $data['messages'],
            'contact'      => $data['contact'],
            'assignee'     => $data['assignee'],
        ]);
    }

    /**
     * AJAX - Polling nouveaux messages
     * Appelé par setInterval() côté frontend
     */
    public function pollMessages(int $conversationId, Request $request): JsonResponse
    {
        $lastMessageId = $request->get('last_message_id', 0);

        $newMessages = $this->conversationService->getNewMessages(
            $conversationId,
            $lastMessageId
        );

        return response()->json([
            'messages' => $newMessages,
```

```
            'has_new'  => count($newMessages) > 0,
        ]);
    }

    /**
     * AJAX - Envoyer un message
     */
    public function sendMessage(int $conversationId, Request $request): JsonResponse
    {
        $request->validate([
            'content'    => 'required|string|max:4096',
            'is_private' => 'sometimes|boolean',
        ]);

        if ($request->boolean('is_private')) {
            $result = $this->messageService->sendPrivateNote(
                $conversationId, $request->content
            );
        } else {
            $result = $this->messageService->sendToCustomer(
                $conversationId, $request->content
            );
        }

        return response()->json($result);
    }

    /**
     * AJAX - Résoudre / Réouvrir
     */
    public function toggleStatus(int $conversationId, Request $request): JsonResponse
    {
        $action = $request->get('action', 'resolve');

        $result = match ($action) {
            'resolve' => $this->conversationService->resolve($conversationId),
            'reopen'  => $this->conversationService->reopen($conversationId),
        };

        return response()->json($result);
    }

    /**
     * AJAX - Assigner à un agent
     */
    public function assign(int $conversationId, Request $request): JsonResponse
```

```
    {
        $request->validate(['agent_id' => 'required|integer']);

        $result = $this->conversationService->assign(
            $conversationId, $request->agent_id
        );

        return response()->json($result);
    }

    /**
     * AJAX - Polling compteurs sidebar
     */
    public function counts(): JsonResponse
    {
        return response()->json(
            $this->conversationService->getCounts()
        );
    }
}
```

DashboardController.php

```
<?php

namespace App\Http\Controllers;

use App\Services\Chatwoot\ReportService;
use Illuminate\Http\Request;

class DashboardController extends Controller
{
    public function __construct(
        private ReportService $reportService
    ) {}

    public function index(Request $request)
    {
        $period = $request->get('period', 'today');

        return view('dashboard.index', [
            'stats'       => $this->reportService->getDashboardStats($period),
            'agents'      => $this->reportService->getAgentPerformance($period),
            'period'      => $period,
        ]);
    }
```

```
}
```

TwilioWebhookController.php — LE HANDOFF

```php
<?php

namespace App\Http\Controllers\Webhook;

use App\Http\Controllers\Controller;
use App\Services\Chatwoot\ChatwootClient;
use Illuminate\Http\Request;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Log;

class TwilioWebhookController extends Controller
{
    public function __construct(
        private ChatwootClient $chatwoot
    ) {}

    /**
     * Webhook appelé par Twilio Studio quand le client demande un agent.
     *
     * Dans Twilio Studio, ajouter un widget "Make HTTP Request" :
     * - Method: POST
     * - URL: https://votre-app.com/api/webhooks/twilio/handoff
     * - Body: {
     *      "from": "{{trigger.message.From}}",
     *      "body": "{{trigger.message.Body}}",
     *      "name": "{{flow.variables.customer_name}}"
     *   }
     */
    public function handoff(Request $request): JsonResponse
    {
        $from = $request->input('from');        // ex: whatsapp:+2250700000000
        $body = $request->input('body', '');     // Dernier message du client
        $name = $request->input('name', 'Client WhatsApp');

        Log::info('Twilio Handoff', compact('from', 'body', 'name'));

        try {
            // 1. Chercher si le contact existe déjà dans Chatwoot
            $phone = str_replace('whatsapp:', '', $from);
            $searchResult = $this->chatwoot->searchContacts($phone);
            $contacts = $searchResult['payload'] ?? [];
```

```
            if (count($contacts) > 0) {
                $contactId = $contacts[0]['id'];
                $sourceId  = $contacts[0]['contact_inboxes'][0]['source_id'] ?? $from;
            } else {
                // 2. Créer le contact
                $newContact = $this->chatwoot->createContact(
                    name: $name,
                    phoneNumber: $phone,
                );
                $contactId = $newContact['payload']['contact']['id'];
                $sourceId  = $newContact['payload']['contact_inbox']['source_id'] ?? $from;
            }

            // 3. Créer la conversation avec le message initial
            $conversation = $this->chatwoot->createConversation(
                sourceId: $sourceId,
                inboxId: config('chatwoot.whatsapp_inbox_id'),
                contactId: $contactId,
                initialMessage: $body ?: "Le client a demandé à parler à un agent."
            );

            Log::info('Conversation créée', [
                'conversation_id' => $conversation['id'] ?? null,
            ]);

            return response()->json([
                'success'         => true,
                'conversation_id' => $conversation['id'] ?? null,
            ]);

        } catch (\Exception $e) {
            Log::error('Erreur Handoff Twilio → Chatwoot', [
                'error' => $e->getMessage(),
            ]);

            return response()->json([
                'success' => false,
                'error'   => $e->getMessage(),
            ], 500);
        }
    }
}
```

`ChatwootWebhookController.php` — **CHATWOOT → TWILIO**

```php
<?php

namespace App\Http\Controllers\Webhook;

use App\Http\Controllers\Controller;
use App\Services\Twilio\TwilioService;
use Illuminate\Http\Request;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Log;

class ChatwootWebhookController extends Controller
{
    public function __construct(
        private TwilioService $twilio
    ) {}

    /**
     * Webhook Chatwoot → Laravel
     *
     * Dans Chatwoot : Settings → Integrations → Webhooks → Ajouter
     * URL: https://votre-app.com/api/webhooks/chatwoot
     * Events: message_created, conversation_status_changed
     */
    public function handle(Request $request): JsonResponse
    {
        $event = $request->input('event');

        Log::info('Chatwoot Webhook', ['event' => $event]);

        return match ($event) {
            'message_created'              => $this->handleNewMessage($request),
            'conversation_status_changed'  => $this->handleStatusChange($request),
            default                        => response()->json(['ok' => true]),
        };
    }

    /**
     * Quand un agent envoie un message dans Chatwoot,
     * on le transmet au client via Twilio WhatsApp
     */
    private function handleNewMessage(Request $request): JsonResponse
    {
        $messageType = $request->input('message_type');
        $private     = $request->input('private', false);
```

```
        // Ne transmettre que les messages sortants (agent → client)
        // et qui ne sont pas des notes privées
        if ($messageType !== 'outgoing' || $private) {
            return response()->json(['ok' => true, 'skipped' => true]);
        }

        $content     = $request->input('content', '');
        $phone       = $request->input('conversation.contact_inbox.source_id')
                    ?? $request->input('conversation.meta.sender.phone_number');

        if (!$phone || !$content) {
            return response()->json(['ok' => true, 'skipped' => 'no_phone_or_content']);
        }

        // Envoyer via Twilio WhatsApp
        $this->twilio->sendWhatsApp($phone, $content);

        return response()->json(['ok' => true, 'sent' => true]);
    }

    /**
     * Quand une conversation est résolue,
     * optionnellement notifier le client
     */
    private function handleStatusChange(Request $request): JsonResponse
    {
        $status = $request->input('status');
        $phone  = $request->input('meta.sender.phone_number');

        if ($status === 'resolved' && $phone) {
            $this->twilio->sendWhatsApp(
                $phone,
                "Merci de nous avoir contacté ! Votre demande a été traitée. N'hésitez pas à
            );
        }

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

6. Service Twilio — `TwilioService.php`

```php
<?php

namespace App\Services\Twilio;

use Twilio\Rest\Client;

class TwilioService
{
    private Client $client;
    private string $fromNumber;

    public function __construct()
    {
        $this->client = new Client(
            config('services.twilio.sid'),
            config('services.twilio.auth_token')
        );
        $this->fromNumber = config('services.twilio.whatsapp_from');
    }

    /**
     * Envoyer un message WhatsApp via Twilio
     */
    public function sendWhatsApp(string $to, string $body): void
    {
        // S'assurer du format whatsapp:+225XXXXXXXXX
        if (!str_starts_with($to, 'whatsapp:')) {
            $to = "whatsapp:{$to}";
        }

        $this->client->messages->create($to, [
            'from' => $this->fromNumber,
            'body' => $body,
        ]);
    }
}
```

---

7. Routes

`routes/web.php`

```php
<?php
```

```php
use App\Http\Controllers\Auth\LoginController;
use App\Http\Controllers\ConversationController;
use App\Http\Controllers\DashboardController;
use App\Http\Controllers\AgentController;

//   Auth
Route::get('/login', [LoginController::class, 'showForm'])->name('login');
Route::post('/login', [LoginController::class, 'login']);
Route::post('/logout', [LoginController::class, 'logout'])->name('logout');

//   App (authentifié)
Route::middleware('auth')->group(function () {

    // Dashboard stats
    Route::get('/', [DashboardController::class, 'index'])->name('dashboard');

    // Conversations
    Route::get('/conversations', [ConversationController::class, 'index'])
        ->name('conversations.index');

    Route::get('/conversations/{conversationId}', [ConversationController::class, 'show'])
        ->name('conversations.show');

    // AJAX - Polling & Actions
    Route::prefix('ajax')->group(function () {
        Route::get('/conversations/{conversationId}/poll', [ConversationController::class,
            ->name('ajax.poll');

        Route::post('/conversations/{conversationId}/messages', [ConversationController::cla
            ->name('ajax.send');

        Route::post('/conversations/{conversationId}/status', [ConversationController::class
            ->name('ajax.status');

        Route::post('/conversations/{conversationId}/assign', [ConversationController::class
            ->name('ajax.assign');

        Route::get('/conversations/counts', [ConversationController::class, 'counts'])
            ->name('ajax.counts');
    });

    // Agents
    Route::get('/agents', [AgentController::class, 'index'])->name('agents.index');
});
```

routes/api.php

```php
<?php

use App\Http\Controllers\Webhook\TwilioWebhookController;
use App\Http\Controllers\Webhook\ChatwootWebhookController;

//   Webhooks (pas d'auth Laravel, auth par signature/secret)
Route::prefix('webhooks')->group(function () {

    // Twilio Studio → Handoff vers Chatwoot
    Route::post('/twilio/handoff', [TwilioWebhookController::class, 'handoff'])
        ->name('webhook.twilio.handoff');

    // Chatwoot → Notification message agent → Twilio WhatsApp
    Route::post('/chatwoot', [ChatwootWebhookController::class, 'handle'])
        ->name('webhook.chatwoot');
});
```

---

8. Polling JavaScript (Frontend)

À inclure dans conversations/show.blade.php :

```javascript
/**
 * Polling - Récupère les nouveaux messages toutes les 4 secondes
 */
document.addEventListener('DOMContentLoaded', function () {
    const conversationId = {{ $conversation->id }};
    const messagesContainer = document.getElementById('messages-container');
    const messageForm = document.getElementById('message-form');
    const messageInput = document.getElementById('message-input');

    let lastMessageId = {{ collect($messages)->max('id') ?? 0 }};
    const POLL_INTERVAL = {{ config('chatwoot.polling_interval', 4000) }};

    //   Polling
    setInterval(async () => {
        try {
            const response = await fetch(
                `/ajax/conversations/${conversationId}/poll?last_message_id=${lastMessageId}
            );
            const data = await response.json();

            if (data.has_new) {
                data.messages.forEach(msg => {
```

```
                appendMessage(msg);
                lastMessageId = Math.max(lastMessageId, msg.id);
            });
            scrollToBottom();
        }
    } catch (error) {
        console.error('Erreur polling:', error);
    }
}, POLL_INTERVAL);

//   Envoi message
messageForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const content = messageInput.value.trim();
    if (!content) return;

    messageInput.value = '';
    messageInput.focus();

    try {
        const response = await fetch(`/ajax/conversations/${conversationId}/messages`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-token"]').conten
            },
            body: JSON.stringify({ content }),
        });
        const data = await response.json();
        appendMessage(data);
        lastMessageId = Math.max(lastMessageId, data.id);
        scrollToBottom();
    } catch (error) {
        console.error('Erreur envoi:', error);
    }
});

//   Helpers
function appendMessage(msg) {
    const isOutgoing = msg.message_type === 1;
    const isPrivate = msg.private;
    const div = document.createElement('div');
    div.className = `message ${isOutgoing ? 'outgoing' : 'incoming'} ${isPrivate ? 'priv
    div.innerHTML = `
        <div class="message-bubble">
            <p>${escapeHtml(msg.content)}</p>
```

```
                <span class="message-time">${formatTime(msg.created_at)}</span>
                ${isPrivate ? '<span class="badge-private">Note privée</span>' : ''}
            </div>
        `;
        messagesContainer.appendChild(div);
    }

    function scrollToBottom() {
        messagesContainer.scrollTop = messagesContainer.scrollHeight;
    }

    function escapeHtml(text) {
        const div = document.createElement('div');
        div.textContent = text;
        return div.innerHTML;
    }

    function formatTime(timestamp) {
        const date = new Date(timestamp * 1000);
        return date.toLocaleTimeString('fr-FR', { hour: '2-digit', minute: '2-digit' });
    }
});
```

---

9. Enums

ConversationStatus.php

```php
<?php

namespace App\Enums;

enum ConversationStatus: string
{
    case OPEN     = 'open';
    case PENDING  = 'pending';
    case RESOLVED = 'resolved';
    case SNOOZED  = 'snoozed';
    case ALL      = 'all';
}
```

MessageType.php

```php
<?php

namespace App\Enums;
```

```php
enum MessageType: int
{
    case INCOMING = 0;  // Message du client
    case OUTGOING = 1;  // Message de l'agent
    case ACTIVITY = 2;  // Activité système (assignation, résolution...)
}
```

---

10. Résumé des endpoints Chatwoot utilisés

Action	Méthode	Endpoint Chatwoot
Lister conversations	`GET`	`/api/v1/accounts/{id}/conversations`
Détail conversation	`GET`	`/api/v1/accounts/{id}/conversations/{conv_id}`
Créer conversation	`POST`	`/api/v1/accounts/{id}/conversations`
Toggle statut	`POST`	`/api/v1/accounts/{id}/conversations/{conv_id}/toggle_statu`
Assigner agent	`POST`	`/api/v1/accounts/{id}/conversations/{conv_id}/assignments`
Lister messages	`GET`	`/api/v1/accounts/{id}/conversations/{conv_id}/messages`
Envoyer message	`POST`	`/api/v1/accounts/{id}/conversations/{conv_id}/messages`
Rechercher contacts	`GET`	`/api/v1/accounts/{id}/contacts/search?q=`
Créer contact	`POST`	`/api/v1/accounts/{id}/contacts`
Lister agents	`GET`	`/api/v1/accounts/{id}/agents`
Rapport agents	`GET`	`/api/v1/accounts/{id}/reports/agents`
Rapport compte	`GET`	`/api/v1/accounts/{id}/reports/account`
Compteurs	`GET`	`/api/v1/accounts/{id}/conversations/meta`

---

11. Ordre d'implémentation recommandé

1. **Semaine 1** — Setup Chatwoot self-hosted + API Channel inbox + `ChatwootClient.php` + tests avec Postman
2. **Semaine 2** — Handoff Twilio → Chatwoot (webhook) + ChatwootWebhook → Twilio (retour message agent)
3. **Semaine 3** — UI conversations (liste + fil de discussion + polling + envoi message)
4. **Semaine 4** — Dashboard stats + gestion agents + polish UI
