

AXYNIA

Gateway Webchat

Especificación técnica de desarrollo

Laravel / PHP + Workerman + MQTT

Decisión central - El Gateway se implementará como un proceso PHP persistente dentro de un proyecto Laravel. Workerman será el motor WebSocket y event loop; Apache seguirá terminando TLS y actuando como reverse proxy.

Campo	Definición
Versión	0.02
Fecha	1 de agosto de 2026
Estado	DRAFT para implementación
Dominio público	gateway.axynia.smart-admin.biz
Endpoint público	wss://gateway.axynia.smart-admin.biz/ws
Escucha interna	websocket://127.0.0.1:3100
Directorio	/var/www/html/gateway.axynia.smart-admin.biz
Runtime	PHP >= 8.1, Laravel, Workerman 5.x
Supervisión	Supervisor
Ubicación prevista del PDF	docs/Axynia_Gateway_Especificacion_Desarrollo_v0.02.pdf

Entrega inmediata - Reproducir el gateway de eco: recibir un mensaje nativo de Deep Chat y responder por el mismo WebSocket con {"role":"ai","text":"Recibido: ..."}. Sin MQTT ni sesiones reales en esta primera etapa.

Arquitectura final - El mismo proceso Workerman publicará mensajes en MQTT y escuchará las respuestas de retorno. No se usará un segundo daemon MQTT en v0.02.

Control documental

Versión	Fecha	Estado	Alcance
0.02	01/08/2026	DRAFT	Especificación integral del Gateway en Laravel/PHP, identificación chatbot_id, eco inicial, protocolo de control, MQTT, resume y escalado horizontal.

Contenido

- 1. 1. Objetivo, principios y alcance
- 2. 2. Arquitectura global
- 3. 3. Plataforma de ejecución: Laravel + Workerman
- 4. 4. Contrato Deep Chat y fase inicial de eco
- 5. 5. Identificación del chatbot mediante UUID
- 6. 6. Contexto y registro de conexiones
- 7. 7. Dispatcher y handlers
- 8. 8. Protocolo Axynia Webchat v1 - DRAFT
- 9. 9. Ciclo de sesión y reanudación
- 10. 10. Integración MQTT - DRAFT
- 11. 11. Escalado horizontal y alta disponibilidad
- 12. 12. Seguridad y límites
- 13. 13. Operación, Supervisor y observabilidad
- 14. 14. Estructura propuesta del proyecto
- 15. 15. Plan de implementación y criterios de aceptación
- 16. 16. Decisiones cerradas y pendientes
- 17. 17. Referencias técnicas

Convenciones

Marca	Significado
REQUERIDO	Condición obligatoria para considerar cumplido el alcance.
RECOMENDADO	Decisión propuesta; puede modificarse documentando el motivo.
DRAFT	Contrato futuro sujeto a aprobación y pruebas end-to-end.
FUERA DE ALCANCE	No debe implementarse en la fase indicada.

Documento autónomo - Esta especificación contiene toda la información necesaria para que el equipo Laravel/PHP inicie el Gateway. Debe incorporarse al repositorio del Gateway, no al árbol del frontend Deep Chat.

1. Objetivo, principios y alcance

El objetivo es implementar un Gateway de transporte para el canal Webchat de Axynia. El componente recibe frames WebSocket desde el navegador, valida y normaliza los eventos, publica comandos o mensajes en MQTT, consume las respuestas del backend y las devuelve al socket correcto.

El Gateway no es un backend de negocio. No administra como fuente de verdad las sesiones, el historial, el contexto del agente, los prompts, las tools ni la persistencia.

1.1 Principios obligatorios

- Proceso PHP persistente; no se utilizará PHP-FPM para mantener conexiones WebSocket.
- Laravel aporta configuración, contenedor, logging, servicios, testing y organización del código.
- Workerman escucha 127.0.0.1:3100 y ejecuta todos los callbacks de red mediante un event loop asíncrono.
- La instancia inicial usa un único worker Workerman: count = 1.
- Quedan prohibidas las operaciones bloqueantes o pesadas dentro del proceso Gateway.
- WebSocket y cliente MQTT conviven en el mismo worker/event loop.
- Apache expone WSS, termina TLS y reenvía a WS local.
- El Gateway conserva solo estado efímero de conexiones activas.
- Las sesiones pueden reanudarse desde cualquier instancia de Gateway.

1.2 Alcance inmediato - Fase A: eco

- Aceptar el handshake en /ws con chatbot_id UUID y Origin permitido.
- Recibir frames de texto con JSON nativo de Deep Chat.
- Extraer el último mensaje textual del usuario.
- Invocar handleMessage(request, connectionContext).
- Responder exactamente "Recibido: " + texto original mediante un Response compatible con Deep Chat.
- Implementar validación, logs, heartbeat, cierre ordenado y ejecución bajo Supervisor.
- No conectar todavía a MQTT ni crear sesiones reales.

1.3 Alcance objetivo - Fases posteriores

- Adaptador frontend mediante Deep Chat connect.handler.
- Eventos de control session.create, session.resume y session.close.
- Publicación y suscripción MQTT asíncrona.
- Correlación gateway_id + connection_id + conversation_id.
- Reanudación con historial completo o sincronización incremental.
- Escalado horizontal con múltiples VMs Gateway detrás de un balanceador.

1.4 Prohibiciones de diseño

Prohibido	Motivo
Consultas SQL o Eloquent en el camino de cada mensaje	Pueden bloquear el event loop y trasladan lógica de backend al Gateway.
HTTP sincrónico, sleep(), esperas activas o llamadas LLM	Detienen la atención de todos los sockets del worker.
Persistir historial como fuente de verdad	Impide que una sesión pueda reanudarse en otra instancia.
Ejecutar tools o reglas de negocio	Responsabilidad de los workers backend.
Exponer el broker MQTT al navegador	Rompe la separación y el control de seguridad.
Múltiples workers Workerman en v0.02	Cada proceso tendría memoria y sockets independientes; requiere IPC/ruteo adicional.

2. Arquitectura global

Arquitectura objetivo del Gateway Webchat

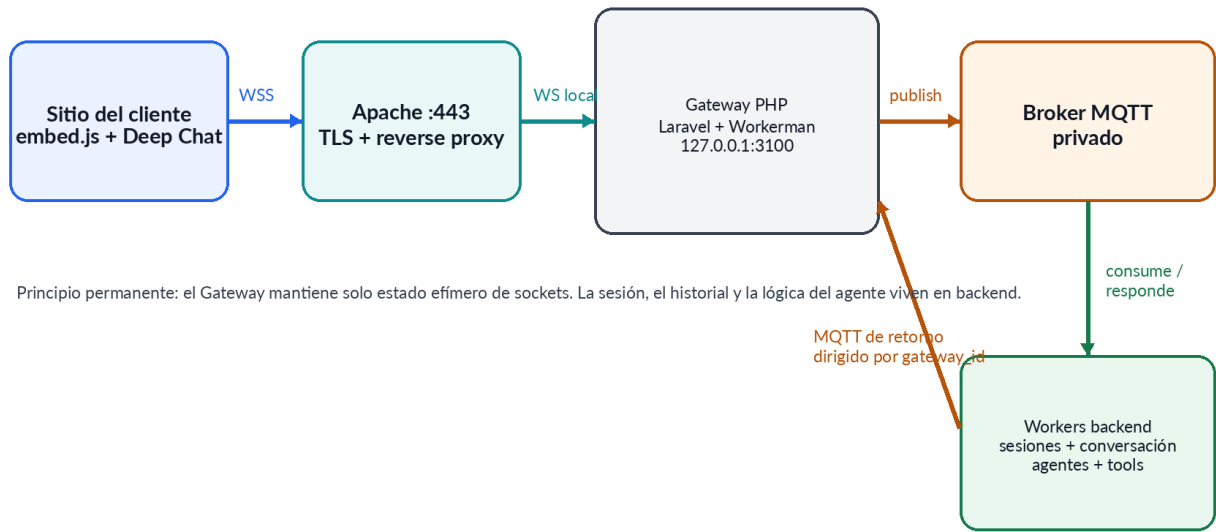


Figura 1. Flujo completo previsto entre frontend, Apache, Gateway, MQTT y workers backend.

2.1 Recorrido de transporte

1. El sitio del cliente carga embed.js desde webchat.axynia.smart-admin.biz e incluye el chatbot_id asignado.
2. El loader crea el launcher y la interfaz Deep Chat.
3. El navegador abre wss://gateway.axynia.smart-admin.biz/ws?chatbot_id=<UUID>.
4. Apache recibe la conexión en 443, termina TLS y hace reverse proxy hacia websocket://127.0.0.1:3100.
5. Workerman acepta la conexión, genera connection_id y registra el contexto efímero.
6. En la fase de eco responde localmente. En la fase MQTT publica el mensaje al backend.
7. El backend publica la respuesta dirigida al gateway_id que posee el socket.
8. El Gateway localiza la conexión y envía el evento o Response al navegador.

2.2 Responsabilidades por componente

Componente	Responsabilidad
embed.js / adaptador	Identificar chatbot, abrir/reconectar WSS, generar eventos Axynia, guardar resume_token y traducir mensajes hacia/desde Deep Chat.
Deep Chat	Renderizar la conversación y entregar/recibir objetos Request message y Response.
Apache	TLS/Certbot, reverse proxy WebSocket, límites perimetrales y logs de acceso.
Gateway Laravel/Workerman	Sockets, validación, despacho, heartbeat, publicación MQTT y ruteo de respuestas.
Session workers	Crear, reanudar, validar y cerrar sesiones; emitir tokens; decidir TTL e historial.
Conversation workers	Persistir mensajes, recuperar contexto, ejecutar agentes/tools y producir respuestas.
Broker MQTT	Transporte privado y desacoplamiento entre Gateway y backend.

Separación crítica - Los objetos WebSocket solo existen en memoria del Gateway. La sesión y la conversación deben sobrevivir a cualquier reinicio o cambio de instancia.

3. Plataforma de ejecución: Laravel + Workerman

Workerman no reemplaza a Laravel ni funciona como una clase propia del framework. Es el motor de red asíncrono que se instala por Composer y se integra dentro del proyecto Laravel. Laravel se inicia una sola vez; luego el proceso permanece vivo atendiendo eventos.

3.1 Dependencias base

Componente	Requisito / criterio
Sistema operativo	Linux/POSIX. Workerman requiere un entorno compatible con procesos y señales.
PHP	Mínimo 8.1 para Workerman 5.x. La versión exacta debe quedar verificada y documentada.
Extensiones	json, pcntl y posix. ext-event es opcional/recomendada para rendimiento, no obligatoria para la primera fase.
Laravel	Versión seleccionada por el proyecto y fijada mediante composer.lock.
Workerman	Rama 5.x, versión exacta fijada en composer.lock.
MQTT	workerman/mqtt 2.x cuando se habilite la fase MQTT.
Supervisor	Ejecuta el proceso en foreground y lo reinicia ante fallas.

Dependencias Composer propuestas

```
composer require workerman/workerman:^5.2
# Se incorpora en la fase MQTT:
composer require workerman/mqtt:^2.2
```

3.2 Punto de entrada

El repositorio tendrá un archivo gateway.php en la raíz. Su responsabilidad será cargar Composer, iniciar Laravel mediante el bootstrap de consola correspondiente a la versión del framework, resolver el servicio GatewayServer desde el contenedor y ejecutar Worker::runAll().

Ciclo conceptual de arranque

```
Supervisor
-> /usr/bin/php gateway.php start
    -> bootstrap Laravel una vez
    -> crear Worker('websocket://127.0.0.1:3100')
    -> configurar callbacks y MQTT
    -> Worker::runAll()
```

No usar daemon interno - Bajo Supervisor no debe ejecutarse Workerman con -d. Supervisor debe conservar el control del proceso maestro y recibir SIGTERM para un cierre ordenado.

3.3 Modelo de proceso y concurrencia

Modelo de ejecución dentro de una instancia

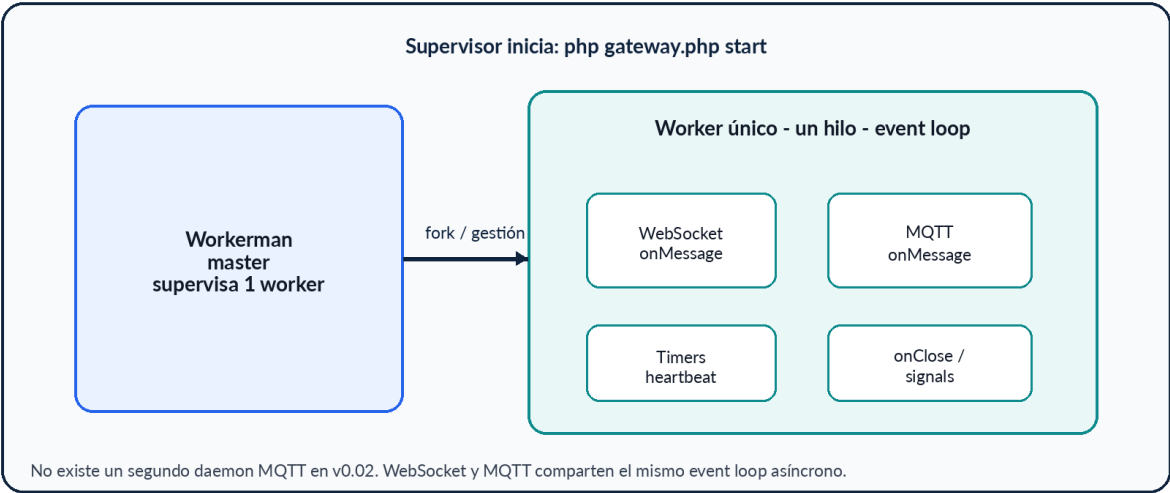


Figura 2. Workerman usa un master y un único worker de aplicación en v0.02.

- El worker de aplicación es monohilo y event-driven.
- La concurrencia proviene de I/O no bloqueante, no de múltiples threads.
- Cada callback debe finalizar rápidamente y devolver el control al event loop.
- WebSocket onMessage, MQTT onMessage, heartbeat y cierres se intercalan dentro del mismo loop.
- Un callback lento bloquea todas las conexiones atendidas por ese worker.

3.4 Regla de no bloqueo

Permitido	No permitido
Parsear y validar JSON	Consultas de base de datos en el mensaje.
Actualizar mapas en memoria	Llamadas HTTP sincrónicas.
Publicar MQTT de forma asíncrona	Invocar Responses API/LLM.
Enviar ACK/Response al socket	sleep(), bucles de espera o archivos grandes.
Logs breves y estructurados	Procesamiento pesado de archivos o imágenes.

4. Contrato Deep Chat y fase inicial de eco

Deep Chat puede conectarse directamente mediante WebSocket. En ese modo envía strings JSON con la forma Request message y espera strings JSON con la forma Response. Los eventos de sesión de Axynia no son eventos nativos de Deep Chat.

4.1 Solicitud nativa de Deep Chat

Deep Chat -> Gateway

```
{
  "messages": [
    {
      "role": "user",
      "text": "Hola"
    }
  ]
}
```

La raíz contiene messages. Para v0.02 solo se admite contenido textual. El frontend debe configurar `requestBodyLimits.maxMessages = 1`; el Gateway soportará defensivamente un array mayor tomando el último mensaje textual con `role="user"`.

4.2 Response esperado por Deep Chat

Gateway -> Deep Chat

```
{
  "role": "ai",
  "text": "Recibido: Hola"
}
```

Deep Chat también contempla la propiedad `error`. En la fase de eco solo se utilizarán `role`, `text` y `error`.

4.3 Comportamiento exacto de la Fase A

1. Aceptar el handshake si `chatbot_id` y `Origin` cumplen las reglas.
2. No enviar `gateway.ready` ni ningún frame espontáneo al abrir la conexión.
3. Recibir un frame de texto UTF-8.
4. Parsear JSON y comprobar que `messages` sea un array.
5. Seleccionar el último elemento con `role="user"` y `text` string no vacío.
6. Invocar `handleNewMessage(request, connectionContext)`.
7. Responder `{"role":"ai","text":"Recibido: " + originalText}`.
8. Mantener el socket abierto salvo violaciones de tamaño, protocolo o seguridad.

4.4 Contrato de `handleNewMessage`

Aspecto	Definición
Firma conceptual	<code>handleNewMessage(array \$request, ConnectionContext \$context): HandlerResult</code>
Entrada	Request message ya parseado y contexto efímero de conexión.
Procesamiento actual	Validación y selección del texto. No hay lógica de negocio.
Salida Fase A	Response Deep Chat con el eco.
Salida futura	Payload normalizado para MQTT; la respuesta llegará por el callback MQTT.
Errores	Resultado tipado; ninguna excepción debe escapar del dispatcher.

4.5 Validaciones mínimas

Condición	Acción
Frame binario	Rechazar; v0.02 solo admite texto JSON.

Condición	Acción
JSON inválido	Responder {"error":"JSON inválido"}.
messages ausente/no array	Responder {"error":"Formato Deep Chat inválido"}.
Sin mensaje user textual	Responder error recuperable.
Texto vacío	Responder error; socket abierto.
html/files/custom	Fuera de alcance; responder error claro.
Exceso de tamaño o abuso	Cerrar con código WebSocket documentado y registrar.

5. Identificación del chatbot mediante UUID

Definición - Cada bot publicable tendrá un chatbot_id persistente de tipo UUID generado por Axynia. El UUID identifica al chatbot, pero es público y no constituye una credencial.

5.1 Código de inserción

Código propuesto para el sitio del cliente

```
<script
  src="https://webchat.axynia.smart-admin.biz/embed.js"
  data-chatbot-id="550e8400-e29b-41d4-a716-446655440000"
  async>
</script>
```

El loader servido por Axynia se ejecutará en la página del cliente, creará el launcher y abrirá el WebSocket. Esta especificación asume inyección directa en el DOM, no iframe.

5.2 Apertura WSS

URL pública

```
wss://gateway.axynia.smart-admin.biz/ws
?chatbot_id=550e8400-e29b-41d4-a716-446655440000
```

El Gateway toma chatbot_id del query string durante el handshake y lo fija como inmutable para toda la vida del socket.

5.3 Definición formal

Campo	Definición
Nombre	chatbot_id
Tipo	UUID persistente
Entidad	Chatbot/configuración publicable
Origen	Código de inserción entregado al cliente
Transporte inicial	Query parameter del handshake WSS
Visibilidad	Público; visible en HTML y tráfico WSS
Uso	Resolver cliente, configuración, agente, trazabilidad, límites y ruteo
Seguridad	No autentica. Debe combinarse con estado del bot, Origin permitido y límites.

5.4 Validación de Origin

Como embed.js se ejecuta dentro de la página del cliente, el navegador enviará como Origin el dominio del sitio cliente. La configuración futura del chatbot debe contener una lista de orígenes permitidos.

Configuración conceptual

```
{
  "chatbot_id": "550e8400-e29b-41d4-a716-446655440000",
  "enabled": true,
  "allowed_origins": [
    "https://cliente.com",
    "https://www.cliente.com"
  ]
}
```

Advertencia - Si en el futuro se utiliza un iframe alojado en Axynia, el Origen observado será el del iframe. Ese diseño requerirá postMessage y, preferentemente, un token de bootstrap firmado.

5.5 Errores mínimos

Código	Condición	Acción
CHATBOT_ID_REQUIRED	No recibido	Rechazar handshake o cerrar según política.
CHATBOT_ID_INVALID	No es UUID	Rechazar antes de procesar frames.
CHATBOT_DISABLED	Inexistente o inhabilitado	Cerrar sin publicar mensajes.
ORIGIN_NOT_ALLOWED	Dominio no autorizado	Rechazar y registrar el evento.
CHATBOT_SESSION_MISMATCH	Token/session de otro chatbot	Rechazar resume y registrar.

6. Contexto y registro de conexiones

Cada socket tendrá un ConnectionContext asociado. El objeto vive únicamente en memoria y se elimina al cerrar la conexión.

Modelo lógico sugerido

```
final class ConnectionContext
{
  public string $connectionId;
  public string $gatewayId;
  public string $chatbotId;
  public ?string $origin;
  public ?string $remoteIp;
  public ?string $userAgent;
  public int $connectedAt;
  public int $lastActivityAt;
  public ?string $sessionId = null;
  public ?string $conversationId = null;
  public ?string $lastMessageId = null;
  public string $state = 'CONNECTED';
}
```

6.1 Identificadores

Identificador	Responsabilidad	Persistencia
chatbot_id	Identifica la configuración publicable del bot.	Backend; permanente.
gateway_id	Identifica la instancia Gateway y dirige respuestas MQTT.	Configuración de instancia.
connection_id	Localiza el socket físico dentro de una instancia.	Solo memoria.
session_id	Identifica el ciclo de sesión.	Backend.
conversation_id	Clave del historial/contexto conversacional.	Backend; índice efímero en Gateway.
request_id	Correlaciona una acción del navegador con ACK/resultado.	Temporal.
event_id	Trazabilidad e idempotencia del evento.	Backend/logs.
message_id	Identidad estable de un mensaje.	Backend y cliente para

Identificador	Responsabilidad	Persistencia
		sincronización.

6.2 Registros en memoria

Mapas mínimos

```

connection_id  -> TcpConnection + ConnectionContext
conversation_id -> connection_id

# opcional y temporal
request_id     -> metadatos de correlación

```

No deben almacenarse en estos mapas objetos Eloquent, Request HTTP, respuestas completas, contenedores de servicio ni estructuras que crezcan sin límite.

6.3 Ciclo de vida

1. Handshake válido: crear connection_id y ConnectionContext.
2. session.created/session.resumed: vincular session_id y conversation_id.
3. message.send: actualizar lastActivityAt y, cuando corresponda, lastMessageId.
4. onClose: eliminar todos los índices asociados al socket.
5. SIGTERM/reinicio: los sockets se cierran; el frontend debe reconectar y reanudar.

Inmutabilidad - chatbot_id, gateway_id y connection_id no cambian durante la vida del socket. session_id y conversation_id se completan después de crear o reanudar la sesión.

7. Dispatcher y handlers

El Gateway debe separar transporte, validación, despacho y handlers. No se debe concentrar todo el comportamiento en el callback `onMessage` de `Workerman`.

7.1 Regla de despacho

Pseudocódigo

```
al recibir frame JSON:
  si frame.messages es un array:
    handleNewMessage(frame, context)
  si frame.type es un string:
    handleControlEvent(frame, context) // habilitado en fase posterior
  si no:
    sendDeepChatError("Formato de mensaje no soportado")
```

Compatibilidad - Durante la fase de eco no se hace `switch(frame.type)` para mensajes nativos de Deep Chat, porque ese campo no existe.

7.2 Componentes sugeridos

Clase / servicio	Responsabilidad
GatewayServer	Construir <code>Workerman Worker</code> , instalar callbacks y controlar el ciclo de vida.
HandshakeValidator	Validar <code>path</code> , <code>chatbot_id</code> , <code>Origin</code> , <code>IP</code> y límites de conexión.
ConnectionRegistry	Crear, buscar y eliminar <code>ConnectionContext</code> y sus índices.
FrameDecoder	Verificar tipo, tamaño, UTF-8 y parsear JSON.
FrameDispatcher	Distinguir contrato nativo Deep Chat y envelope Axynia.
NewMessageHandler	Procesar entrada textual; eco inicial o publicación MQTT futura.
ControlEventHandler	Crear/reanudar/cerrar sesión cuando se habilite el protocolo.
DeepChatResponseFactory	Construir <code>Response</code> visuales válidos.
AxyniaEventFactory	Construir envelopes versionados.
MqttBridge	Conectar, publicar, suscribir, reconectar y despachar respuestas.

7.3 Excepciones y errores

- Ninguna excepción debe escapar del callback principal de `Workerman`.
- Los errores esperables deben representarse mediante resultados o excepciones tipadas.
- Los errores visuales actuales se convierten a `{"error": "..."}.`
- Los errores futuros se envían como evento Axynia `type="error"` y el adaptador decide si los muestra.
- No se envían stack traces, rutas, credenciales ni detalles MQTT al navegador.

8. Protocolo Axynia Webchat v1 - DRAFT

Aclaración - Los siguientes eventos no son generados nativamente por Deep Chat. Serán producidos e interpretados por embed.js mediante connect.handler.

8.1 Envelope común

Axynia Webchat Protocol v1

```
{
  "v": 1,
  "type": "message.send",
  "request_id": "uuid",
  "event_id": "uuid",
  "sent_at": "2026-08-01T20:00:00.000Z",
  "payload": {}
}
```

Campo	Uso
v	Versión del protocolo JSON.
type	Evento para el dispatcher.
request_id	Correlación de una acción iniciada por cliente.
event_id	Identidad del evento para trazabilidad/idempotencia.
sent_at	Marca UTC ISO-8601 informativa.
payload	Contenido específico.

8.2 Catálogo de eventos

Evento	Dirección	Finalidad
gateway.ready	Gateway -> adapter	Confirma conexión y entrega IDs/heartbeat.
session.create	Adapter -> Gateway	Solicita nueva sesión para el chatbot.
session.created	Gateway -> adapter	Devuelve session_id, conversation_id y resume_token.
session.resume	Adapter -> Gateway	Intenta recuperar una sesión mediante token opaco.
session.resumed	Gateway -> adapter	Confirma reanudación y devuelve sincronización.
session.resume_rejected	Gateway -> adapter	Indica token inválido, vencido o incompatible.
session.close	Adapter -> Gateway	Solicita cierre lógico.
session.closed	Gateway -> adapter	Confirma cierre.
message.send	Adapter -> Gateway	Envía mensaje nuevo del usuario.
message.accepted	Gateway -> adapter	Confirma aceptación/ruteo, no respuesta del agente.
assistant.message	Gateway -> adapter	Entrega una respuesta visualizable.
error	Ambos sentidos	Comunica error de protocolo, sesión o procesamiento.

8.3 Conexión y creación

Gateway -> adapter

```
{
  "v": 1,
  "type": "gateway.ready",
  "event_id": "81dfb3d0-2cf8-4bb7-ae59-f5ad847f07ef",
  "payload": {
    "connection_id": "f98cc91d-fcab-4d83-a003-0ba05edb486b",
    "gateway_id": "gateway-01",
    "heartbeat_interval_ms": 25000
  }
}
```

Adapter -> Gateway

```
{
  "v": 1,
  "type": "session.create",
  "request_id": "4d4c8b16-30b8-4bb5-87c9-79e234f8675f",
  "payload": {
    "chatbot_id": "550e8400-e29b-41d4-a716-446655440000",
    "channel": "webchat",
    "client_instance_id": "uuid-persistido-en-navegador"
  }
}
```

Gateway -> adapter

```
{
  "v": 1,
  "type": "session.created",
  "request_id": "4d4c8b16-30b8-4bb5-87c9-79e234f8675f",
  "event_id": "f143fb24-d15f-4cc3-a3d4-58be50ab1cd8",
  "payload": {
    "session_id": "ses_01...",
    "conversation_id": "conv_01...",
    "resume_token": "token-opaco"
  }
}
```

El chatbot_id de session.create debe coincidir con el chatbot_id asociado al handshake. Después de session.created, los mensajes pueden omitirlo porque el socket ya está vinculado.

8.4 Mensajes conversacionales

Adapter -> Gateway

```
{
  "v": 1,
  "type": "message.send",
  "request_id": "cf518e33-d987-4557-a54a-7f4f913725ea",
  "payload": {
    "message": {
      "role": "user",
      "text": "Necesito conocer el estado de mi servicio"
    }
  }
}
```

Gateway -> adapter

```
{
  "v": 1,
  "type": "message.accepted",
  "request_id": "cf518e33-d987-4557-a54a-7f4f913725ea",
  "event_id": "f2cdb3e1-9ed9-474a-b106-84a29bd19fa5",
  "payload": {
    "message_id": "msg_01...",
    "status": "accepted"
  }
}
```

Gateway -> adapter

```
{
  "v": 1,
  "type": "assistant.message",
  "request_id": "cf518e33-d987-4557-a54a-7f4f913725ea",
  "event_id": "06edff96-f077-4d68-b27c-e28cf87bf60a",
  "payload": {
    "message": {
      "message_id": "msg_02...",
      "role": "ai",
      "text": "Voy a consultar el estado de tu servicio."
    }
  }
}
```

El adaptador no entrega el envelope completo a Deep Chat. Debe invocar conceptualmente `signals.onResponse(event.payload.message)`.

8.5 Error futuro

Evento de error

```
{
  "v": 1,
  "type": "error",
  "request_id": "cf518e33-d987-4557-a54a-7f4f913725ea",
  "event_id": "9e30f13f-22cc-45dc-81ea-4d274e7b972c",
  "payload": {
    "code": "SESSION_NOT_READY",
    "message": "La sesión aún no está disponible.",
    "retryable": true
  }
}
```

8.6 Máquina de estados mínima

Estado	Eventos válidos	Transición
CONNECTED	session.create, session.resume	SESSION_PENDING
SESSION_PENDING	session.created, session.resumed, session.resume_rejected, error	READY o CONNECTED
READY	message.send, session.close	READY o CLOSING
CLOSING	session.closed, error	CLOSED
CLOSED	Ninguno	Cerrar socket o abrir nueva conexión

9. Ciclo de sesión y reanudación

Reanudación de sesión: tres escenarios

A. Corte de red con la página todavía abierta

- El HTML ya conserva el hilo visible.
- Enviar session.resume con restore_history=false y last_message_id.
- Backend devuelve solo missed_messages posteriores a last_message_id.

B. Recarga o reapertura reciente

- El historial renderizado pudo perderse.
- Enviar session.resume con restore_history=true.
- Backend devuelve sesión y últimos N mensajes definidos por política.

C. Sesión vencida o inválida

- Backend responde session.resume_rejected con SESSION_EXPIRED.
- Frontend elimina el token anterior y ejecuta session.create.
- El cliente no decide el TTL: lo decide el backend.

Figura 3. El backend decide si reanuda, sincroniza o crea una nueva sesión.

9.1 Token de reanudación

- El navegador debe guardar un resume_token opaco, no confiar solo en session_id.
- El token se asocia a chatbot_id, session_id, conversation_id y vencimiento.
- Debe ser difícil de adivinar, revocable y preferentemente rotado en cada resume.
- El backend valida que el token pertenezca al chatbot_id del handshake.
- La ubicación de almacenamiento queda desacoplada del Gateway. Recomendación inicial: localStorage con clave por chatbot_id; una cookie first-party puede evaluarse por política del frontend.

Clave recomendada en almacenamiento del navegador

```
axynia:<chatbot_id>:resume_token
```

9.2 Corte de red con página activa

Adapter -> Gateway

```
{
  "v": 1,
  "type": "session.resume",
  "request_id": "uuid",
  "payload": {
    "resume_token": "token-opaco",
    "restore_history": false,
    "last_message_id": "msg_9823"
  }
}
```

El backend no reenvía el hilo completo. Devuelve como máximo los mensajes posteriores a last_message_id para evitar perder una respuesta producida mientras el socket estuvo desconectado.

Sincronización incremental

```
{
  "v": 1,
  "type": "session.resumed",
  "request_id": "uuid",
  "payload": {
    "session_id": "ses_123",
    "conversation_id": "conv_456",
    "resume_token": "token-rotado",
    "missed_messages": [
      {
        "message_id": "msg_9824",
        "role": "ai",
        "text": "Respuesta recibida durante la desconexión."
      }
    ]
  }
}
```

9.3 Recarga o reapertura

Solicitud de reconstrucción

```
{
  "v": 1,
  "type": "session.resume",
  "request_id": "uuid",
  "payload": {
    "resume_token": "token-opaco",
    "restore_history": true
  }
}
```

Si la sesión es reciente, el backend devuelve los últimos N mensajes. N y la ventana de retención son políticas del backend, no parámetros libres del navegador.

9.4 Sesión vencida

Backend -> adapter

```
{
  "v": 1,
  "type": "session.resume_rejected",
  "request_id": "uuid",
  "payload": {
    "code": "SESSION_EXPIRED",
    "retryable": false
  }
}
```

El adaptador elimina el token anterior y ejecuta session.create. El frontend siempre puede intentar resume si encuentra token; la decisión de vigencia pertenece al backend.

10. Integración MQTT - DRAFT

Cuando se habilite MQTT, el mismo worker Workerman mantendrá una conexión asíncrona al broker, publicará mensajes de entrada y se suscribirá a los topics de retorno. No habrá un segundo proceso listener en v0.02.

10.1 Ciclo dentro del event loop

1. onWorkerStart crea MqttBridge y conecta al broker.
2. onConnect se suscribe a topics de retorno de gateway_id.
3. WebSocket onMessage valida y publica comandos/mensajes.
4. MQTT onMessage decodifica la respuesta y busca connection_id/conversation_id.
5. El Gateway envía la respuesta al socket y retorna inmediatamente al loop.
6. onClose/onError aplica reconexión y resuscripción sin bloquear.

10.2 Topics preliminares

Nomenclatura DRAFT

```
# Control generado por gateways
axynia/channels/control/session/create
axynia/channels/control/session/resume
axynia/channels/control/session/close

# Respuestas de control dirigidas
axynia/channels/control/out/{gateway_id}/{connection_id}

# Mensajes conversacionales
axynia/channels/messages/in/{shard}/{conversation_id}
axynia/channels/messages/out/{gateway_id}/{conversation_id}
```

10.3 Payload hacia worker conversacional

Gateway -> MQTT

```
{
  "schema": "axynia.conversation.message.v1",
  "event_id": "uuid",
  "occurred_at": "2026-08-01T20:01:00.000Z",
  "gateway_id": "gateway-01",
  "connection_id": "uuid-efimero",
  "chatbot_id": "550e8400-e29b-41d4-a716-446655440000",
  "session_id": "ses_01...",
  "conversation_id": "conv_01...",
  "request_id": "uuid-del-cliente",
  "channel": "webchat",
  "message": {
    "message_id": "msg_01...",
    "role": "user",
    "text": "Necesito conocer el estado de mi servicio"
  }
}
```

10.4 Respuesta desde backend

Worker -> MQTT -> Gateway

```
{
  "schema": "axynia.conversation.response.v1",
  "event_id": "uuid",
  "gateway_id": "gateway-01",
  "connection_id": "uuid-efimero",
  "conversation_id": "conv_01...",
  "request_id": "uuid-del-cliente",
  "message": {
    "message_id": "msg_02...",
    "role": "ai",
    "text": "Respuesta generada por el agente"
  }
}
```

10.5 Reglas MQTT

- gateway_id debe ser único por instancia lógica y aparecer en el topic/payload de retorno.
- connection_id identifica el socket físico y evita búsquedas ambiguas.
- conversation_id es la clave principal del historial/contexto en backend.
- chatbot_id viaja en cada payload para validación, auditoría, métricas y resolución de configuración.
- QoS, retención y expiración quedan DRAFT; para QoS 1 se requiere idempotencia por event_id/message_id.
- El Gateway no debe publicar retained messages conversacionales.
- Las credenciales MQTT deben estar fuera del código y restringidas por ACL.

11. Escalado horizontal y alta disponibilidad

Escalado horizontal y direccionamiento de respuestas

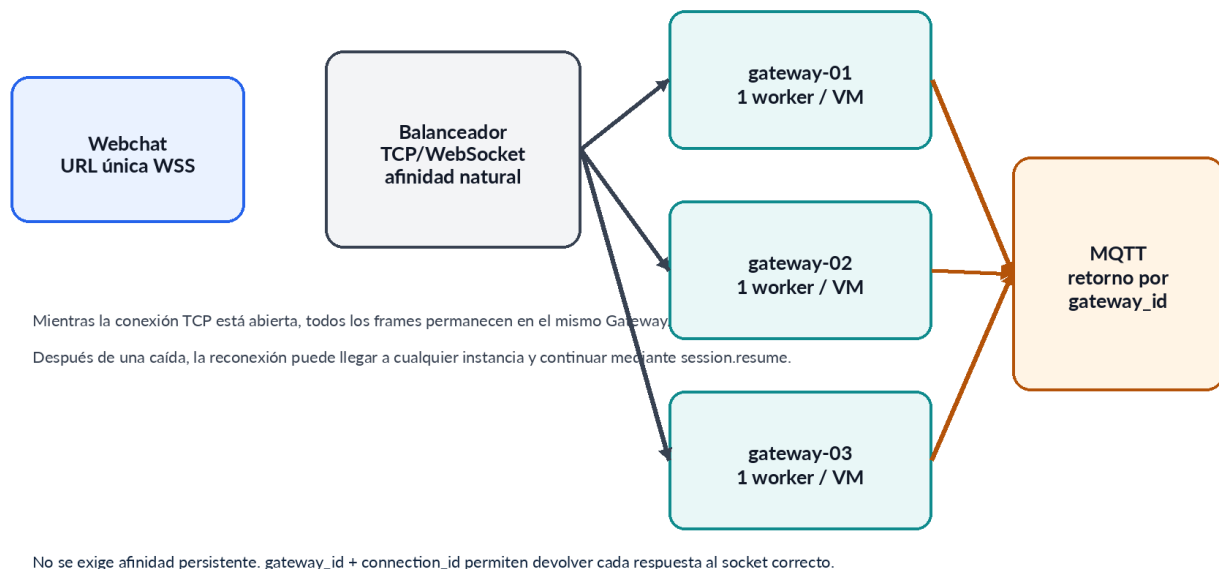


Figura 4. Una URL pública, múltiples instancias reemplazables y retorno MQTT dirigido.

11.1 Afinidad natural

Una conexión WebSocket es una sesión TCP persistente. Una vez que el balanceador asigna el handshake a una instancia, todos los frames de esa conexión continúan en la misma instancia. No se necesita una cookie de afinidad para el socket activo.

11.2 Reconexión sin afinidad persistente

Si el socket se corta, la nueva conexión puede llegar a cualquier Gateway. El frontend ejecuta session.resume y el backend reconstruye el contexto. Esto evita que una sesión dependa de una VM específica.

11.3 Identificación de instancia

Variable obligatoria

GATEWAY_ID=gateway-01

Cada instancia se suscribe solo a sus topics de retorno. La respuesta del backend debe contener gateway_id y connection_id de la conexión que originó el comando.

11.4 Estrategia de despliegue

Etapas	Topología
Inicial	1 VM, 1 proceso Workerman, count=1, 1 conexión MQTT.
Primera escala	2 o más VMs pequeñas detrás de balanceador, cada una con gateway_id único.

Etaa	Topología
Alta disponibilidad	Health checks, retiro ordenado del pool y reconexión/resume en otra instancia.
Aprovechar varios núcleos en una VM	Fuera de v0.02. Requeriría múltiples instancias/puertos con IDs únicos o IPC explícito.

Diseño recomendado - Escalar horizontalmente con VMs reemplazables. No usar estado compartido para sockets; la continuidad se resuelve con session.resume.

12. Seguridad y límites

12.1 Defensa por capas

Capa	Controles
Apache / balanceador	TLS, certificados, límites de handshake/headers, conexiones por IP, timeouts, rate limit perimetral y logs de acceso.
Gateway	Tamaño de frame, JSON válido, longitud de texto, frecuencia por socket/IP/chatbot, Origin, UUID y estados de protocolo.
Backend	Existencia/estado del chatbot, cuota por cliente, autorización de sesión, TTL, idempotencia y políticas de negocio.

12.2 Límites iniciales propuestos

Límite	Valor inicial DRAFT
Frame WebSocket	64 KiB
Mensaje textual	8.000 caracteres
Heartbeat	Ping nativo cada 25 s
Socket sin pong	Cerrar si no respondió al ciclo anterior
Frames binarios	No admitidos
Profundidad JSON	Límite conservador documentado en implementación
Mensajes por segundo	Definir por conexión y chatbot antes de producción

12.3 chatbot_id no es secreto

El UUID puede copiarse desde el HTML del cliente. No debe utilizarse como contraseña, firma ni token de autorización. La autorización se apoya en chatbot habilitado, Origin permitido, límites y, si se adopta, un token de bootstrap firmado.

12.4 Política de cierre

Tipo de error	Acción
Contenido recuperable	Responder Response.error o evento error; socket abierto.
JSON inválido aislado	Responder error; contabilizar.
Frame binario	Rechazar/cerrar según política.
Exceso de tamaño	Cerrar con código WebSocket apropiado.
Origin/chatbot inválido	Rechazar handshake o cerrar inmediatamente.
Abuso/repetición de protocolo inválido	Cerrar, registrar y aplicar rate limit.

12.5 Privacidad de logs

- Registrar IDs y metadatos técnicos, no el contenido completo de mensajes en producción.
- Si se requiere contenido para diagnóstico, truncarlo y habilitarlo mediante configuración temporal.
- Nunca registrar resume_token, passwords MQTT, API keys o cookies completas.
- Aplicar retención y rotación de logs.

13. Operación, Supervisor y observabilidad

13.1 Supervisor

Configuración conceptual

```
[program:axynia-gateway]
directory=/var/www/html/gateway.axynia.smart-admin.biz
command=/usr/bin/php gateway.php start
user=axynia-gateway
autostart=true
autorestart=true
startsecs=3
stopsignal=TERM
stopasgroup=true
killasgroup=true
stopwaitsecs=20
redirect_stderr=true
stdout_logfile=/var/log/axynia-gateway/gateway.log
```

La ruta exacta de PHP, logs y usuario deben verificarse en el servidor. El proceso no debe ejecutarse como root.

13.2 Cierre ordenado

1. Recibir SIGTERM/SIGINT.
2. Dejar de aceptar conexiones nuevas.
3. Marcar la instancia como no saludable o retirarla del balanceador.
4. Cerrar MQTT de manera normal cuando sea posible.
5. Cerrar sockets con código de reinicio/servicio y permitir que el frontend reconecte.
6. Liberar registros de conexiones y finalizar dentro de stopwaitsecs.

13.3 Heartbeat

El Gateway enviará ping WebSocket nativo cada 25 segundos. Si una conexión no respondió pong al ciclo anterior, se cerrará. El heartbeat detecta clientes desaparecidos por redes móviles, CGNAT, NAT y cortes silenciosos.

13.4 Logs estructurados

Ejemplo conceptual

```
{
  "level": "info",
  "event": "ws.message.received",
  "gateway_id": "gateway-01",
  "connection_id": "uuid",
  "chatbot_id": "uuid",
  "conversation_id": "conv_01",
  "bytes": 142,
  "duration_ms": 2
}
```

13.5 Métricas mínimas

- Conexiones activas, aperturas y cierres por motivo.
- Mensajes válidos, inválidos y descartados.
- Latencia de eco y, luego, latencia WS -> MQTT -> WS.
- Estado de conexión MQTT, reconexiones y publicaciones fallidas.
- Uso de memoria y crecimiento de registros en memoria.
- Rate limits aplicados por IP y chatbot_id.

13.6 Variables de entorno propuestas

Configuración base

```
GATEWAY_ID=gateway-01
GATEWAY_HOST=127.0.0.1
GATEWAY_PORT=3100
GATEWAY_PATH=/ws
GATEWAY_WORKERS=1
HEARTBEAT_INTERVAL_MS=25000
MAX_FRAME_BYTES=65536
MAX_MESSAGE_CHARS=8000

MQTT_HOST=broker.internal
MQTT_PORT=1883
MQTT_USERNAME=...
MQTT_PASSWORD=...
MQTT_CLIENT_ID=axynia-gateway-01
MQTT_QOS=1
MQTT_TLS=false
```

Proceso persistente - Cambios en .env o en el código requieren reiniciar/reloadar el Gateway. Laravel no vuelve a bootstrapearse por cada mensaje.

14. Estructura propuesta del proyecto

Árbol recomendado

```
/var/www/html/gateway.axynia.smart-admin.biz
├── app/
│   ├── Gateway/
│   │   ├── Contracts/
│   │   ├── DTO/
│   │   ├── Handlers/
│   │   │   ├── NewMessageHandler.php
│   │   │   └── ControlEventHandler.php
│   │   ├── Mqtt/
│   │   │   └── MqttBridge.php
│   │   ├── Protocol/
│   │   │   ├── FrameDecoder.php
│   │   │   ├── FrameDispatcher.php
│   │   │   └── EventFactory.php
│   │   ├── Security/
│   │   │   └── HandshakeValidator.php
│   │   ├── Support/
│   │   │   ├── ConnectionContext.php
│   │   │   └── ConnectionRegistry.php
│   │   └── GatewayServer.php
│   ├── config/gateway.php
│   ├── docs/
│   │   └── Axynia_Gateway_Especificacion_Desarrollo_v0.02.pdf
│   ├── tests/
│   │   ├── Unit/Gateway/
│   │   └── Integration/Gateway/
│   ├── gateway.php
│   ├── composer.json
│   └── composer.lock
```

14.1 Reglas de implementación

- Usar declare(strict_types=1) en clases del Gateway.
- Preferir DTOs y enums para eventos/estados; evitar arrays sin contrato en capas internas.
- Mantener callbacks Workerman pequeños y delegar en servicios resueltos al inicio.
- No resolver el contenedor completo repetidamente en cada frame si puede evitarse.
- No usar facades o servicios request-scoped que dependan del ciclo HTTP tradicional.
- Liberar referencias a TcpConnection en onClose.
- Fijar dependencias en composer.lock y documentar la versión PHP efectiva.
- Agregar JSON Schema y contract tests antes de habilitar producción MQTT/control.

14.2 Pruebas

Tipo	Cobertura
Unitarias	UUID, Origin, FrameDecoder, selección del último user text, factories, máquina de estados.
Integración local	Cliente WS contra 127.0.0.1:3100; eco, errores, heartbeat y cierre.
End-to-end	WSS contra Apache con certificado válido y chatbot_id.
MQTT	Broker de prueba; publicación, suscripción, reconexión y ruteo por gateway_id.
Resume	Corte con página activa, reapertura reciente, token vencido y mismatch de chatbot.
Carga	Conexiones concurrentes y mensajes cortos, verificando latencia/event loop.

15. Plan de implementación y criterios de aceptación

15.1 Fase A - Gateway de eco Laravel/Workerman

- [] Proyecto Laravel funcional con gateway.php.
- [] workerman/workerman instalado y versionado.
- [] Escucha exclusiva en 127.0.0.1:3100, path /ws.
- [] Handshake valida chatbot_id UUID y Origin.
- [] ConnectionContext y connection_id se crean y destruyen correctamente.
- [] Mensaje Deep Chat produce exactamente "Recibido: " + texto original.
- [] No se envía gateway.ready en modo nativo.
- [] JSON inválido y formatos no soportados generan Response.error.
- [] Heartbeat ping/pong y cierre de sockets muertos.
- [] Supervisor reinicia el proceso y lo ejecuta como axynia-gateway.
- [] Prueba WSS completa a través de Apache exitosa.
- [] Sin MQTT, DB, HTTP privado ni lógica de agentes.

15.2 Fase B - Adaptador y control de sesión

- [] Deep Chat usa connect.handler.
- [] gateway.ready, session.create, session.resume y session.close implementados.
- [] chatbot_id del evento coincide con el handshake.
- [] Estado CONNECTED/SESSION_PENDING/READY/CLOSING/CLOSED validado.
- [] resume_token almacenado y rotado según respuesta backend.

15.3 Fase C - MQTT

- [] workerman/mqtt conectado dentro del mismo event loop.
- [] Publicación de control y conversación sin operaciones bloqueantes.
- [] Suscripción de retorno por gateway_id.
- [] Ruteo a connection_id/conversation_id correcto.
- [] Reconexión MQTT y resuscripción probadas.
- [] Idempotencia definida para QoS adoptado.

15.4 Fase D - Resume y multiinstancia

- [] Resume incremental con last_message_id.
- [] Reapertura reconstruye últimos N mensajes.
- [] Sesión expirada crea una nueva sesión.
- [] Dos Gateways con IDs diferentes reciben conexiones detrás de balanceador.
- [] Una reconexión puede caer en otra VM y continuar la conversación.

15.5 Matriz de pruebas obligatorias Fase A

Prueba	Entrada	Resultado esperado
Eco simple	Hola	Recibido: Hola
Unicode	Acentos, emoji y japonés	Contenido preservado en UTF-8.
Vacío	Solo espacios	Response.error; socket abierto.

Prueba	Entrada	Resultado esperado
JSON roto	Frame no parseable	Response.error o cierre según política.
Binario	Frame binario	Rechazo explícito.
UUID ausente	Sin chatbot_id	Handshake rechazado.
UUID inválido	chatbot_id no UUID	Handshake rechazado.
Origin no permitido	Dominio no autorizado	Handshake rechazado.
Reinicio	Supervisor restart	Cliente detecta cierre y el proceso vuelve a escuchar.
Heartbeat	Cliente sin pong	Socket cerrado al ciclo previsto.

16. Decisiones cerradas y pendientes

16.1 Decisiones cerradas

Tema	Decisión
Stack	Laravel/PHP + Workerman; MQTT mediante workerman/mqtt.
Entrada	gateway.php ejecutado por Supervisor en foreground.
Escucha	127.0.0.1:3100; Apache expone WSS.
Concurrencia	Un worker monohilo con event loop asíncrono.
MQTT	Mismo proceso/event loop; no segundo daemon en v0.02.
Canal WebSocket	Un único socket por instancia de webchat; control y conversación multiplexados.
Identificador bot	chatbot_id UUID público en query WSS y session.create.
Estado	Solo efímero en Gateway; backend es fuente de verdad.
Resume	Cualquier Gateway puede reanudar; no hay afinidad persistente.
Escala	Horizontal mediante múltiples VMs con gateway_id único.

16.2 Pendientes antes de producción

Tema	Decisión pendiente
MQTT	Topics definitivos, QoS, expiración, retención, ACL y shared subscriptions/sharding.
Sesiones	TTL de sesión reciente, rotación/revocación del token y cantidad N de historial.
Frontend	localStorage vs cookie first-party y comportamiento exacto de client_instance_id.
Seguridad	Límites finales por IP/chatbot, origen dinámico y posible token bootstrap firmado.
Mensajes	Política de múltiples mensajes simultáneos por conversación y orden/cancelación.
Errores	Códigos WebSocket y catálogo definitivo de errores Axynia.
Infraestructura	Tecnología de balanceador, health check y procedimiento de rolling restart.
Archivos/streaming	Especificación separada; fuera del contrato textual v0.02.

Criterio de avance - La Fase A debe quedar estable y probada por WSS completo antes de habilitar control de sesión o MQTT. Las fases posteriores deben conservar la separación de responsabilidades definida en este documento.

17. Referencias técnicas

- [Workerman Official Manual - Worker Class](#)
- [Workerman Official Manual - WebSocket Protocol](#)
- [Workerman Official Manual - runAll](#)
- [Workerman Official Manual - workerman/mqtt](#)
- [Deep Chat Official Documentation - Connect](#)
- [Laravel Official Documentation - Artisan Console](#)
- [Packagist - workerman/workerman](#)
- [Packagist - workerman/mqtt](#)

17.1 Vigencia

Las versiones concretas de PHP, Laravel, Workerman y workerman/mqtt deben registrarse en composer.lock y en el README del repositorio. Antes de implementar archivos, streaming, coroutines o múltiples workers por instancia, el equipo debe verificar nuevamente las interfaces oficiales y actualizar esta especificación.

17.2 Registro de revisión

Responsabilidad	Nombre	Fecha	Estado
Desarrollo Gateway Laravel/PHP	Pendiente		
Arquitectura Backend / MQTT	Pendiente		
Frontend Webchat / Deep Chat	Pendiente		
Infraestructura Apache / Balanceador	Pendiente		
Aprobación técnica final	Pendiente		

Próximo hito - Implementar la Fase A en Laravel/Workerman, validar el eco local y repetir la prueba completa WSS -> Apache -> WS -> Gateway PHP -> Deep Chat.

FIN DE LA ESPECIFICACIÓN - VERSIÓN 0.02