C.A.R.L. - Technical Design Document

Communication Assistant for Routing & Logistics

Version: 2.0
Status: Production-Ready Specification
Last Updated: November 29, 2025
Author: Senior Software Architect & DevOps Engineer
Target Stack: Node.js v18+, WhatsApp-Web.js, Docker, OpenAI GPT-4o, Google Calendar API

Executive Summary

C.A.R.L. is a production-grade personal assistant system that operates as a WhatsApp user bot. Unlike traditional business chatbots, C.A.R.L. serves as an executive virtual secretary for individuals suffering from information overload. The system leverages event-driven architecture, AI-powered semantic analysis, and calendar integration to filter noise, manage scheduling, and prepare contextual responses.

Core Philosophy: "Local First, Privacy Focused, AI Powered."

Primary Technical Challenges:

- Session persistence in ephemeral container environments
- Anti-ban mechanisms for WhatsApp's Terms of Service compliance
- Chromium headless browser orchestration in Linux containerized environments
- Cost-optimized OpenAI API usage with context management
- Service Account authentication for unattended calendar operations

---

1. System Architecture

1.1 Event-Driven Architecture Overview

C.A.R.L. operates on a **reactive event loop** powered by Node.js asynchronous I/O model. The system processes WhatsApp messages through a pipeline architecture that ensures resilience, rate limiting, and human-like behavior simulation.

1.2 Message Lifecycle (The Critical Path)

1. **Reception Layer** - Client.on('message') event triggered by Puppeteer-controlled Chromium intercepting WhatsApp WebSocket packets
2. **Gatekeeper Layer** - Multi-stage filtering system:
    - Group message detection (configurable ignore/process)
    - Status/Story message filtering (always ignore)
    - Blacklist/Whitelist verification
    - Rate limiting enforcement
3. **Queue Layer** - Messages enter a ProcessingQueue (never processed synchronously)

4. **Context Enrichment** - Retrieves last N messages from SQLite for conversation continuity
5. **AI Decision Engine** - OpenAI Structured Output analysis produces JSON schema:
6. **Executor Layer** - Simulates human behavior:
    - `chat.sendStateTyping()` activation
    - Random delay: `Math.random() * 3000 + 2000` milliseconds
    - Message transmission
    - State cleanup

1.3 Concurrency Model

Node.js single-threaded event loop handles I/O-bound operations (network calls to OpenAI, Google APIs, WhatsApp WebSocket) efficiently. CPU-bound tasks (message parsing, sanitization) are minimal and synchronous.

Critical Concurrency Constraint: The system MUST NOT process multiple messages from the same sender simultaneously. Solution: Queue implementation with sender-based locking mechanism using `p-queue` library with per-sender concurrency limit of 1.

---

2. Technology Stack Justification

Component	Technology	Technical Justification
Runtime	Node.js v18+	ES Module native support, top-level await, native fetch API, optimal for I/O-bound async operations
WhatsApp Interface	whatsapp-web.js v1.23+	Only mature library supporting WhatsApp Web protocol emulation via Puppeteer, enables personal account usage without Business API costs
Browser Engine	Puppeteer + Chromium	Required by whatsapp-web.js for WebSocket interception and DOM manipulation
AI Processing	OpenAI GPT-4o	Structured Output mode (JSON Schema enforcement), superior intent classification, cost-effective vs GPT-4 Turbo
Calendar Integration	Google Calendar API v3	Industry standard, service account support for unattended operations
Admin Notifications	Telegram Bot API	Zero-setup push notifications, robust webhook alternative, separate from WhatsApp to avoid pollution
Persistence	SQLite3	Embedded database, zero-config, sufficient for <100k messages/year, file-based persistence compatible with Docker volumes
Queue Management	p-queue v7+	Promise-based queue with concurrency control, per-sender rate limiting
Containerization	Docker + Multi-stage builds	Puppeteer dependency isolation, reproducible builds, Render.com compatibility
Hosting	Render.com	Native Docker support, persistent disk volumes, affordable ($7/month with disk), auto-deploy from Git

Table 1: Technology stack with engineering rationale

---

3. Project Structure (Implementation-Ready)

3.1 Directory Tree

```
carl-bot/
├── config/
│   ├── default.json # Non-sensitive config (delays, thresholds)
│   └── tokens.js # Environment variable loader with validation
│
├── src/
│   ├── ai/
│   │   ├── openai.js # OpenAI API wrapper with retry logic
│   │   ├── prompts.js # System prompts & persona definition
│   │   └── structuredOutput.js # JSON Schema definitions for AI responses
│   │
│   ├── services/
│   │   ├── whatsapp.js # WhatsApp client initialization & event handlers
│   │   ├── calendar.js # Google Calendar service account operations
│   │   ├── telegram.js # Telegram admin bot notifications
│   │   └── queue.js # Message processing queue with rate limiting
│   │
│   ├── middleware/
│   │   ├── gatekeeper.js # Message filtering logic (blacklist/whitelist)
│   │   ├── contextBuilder.js # Conversation history retrieval
│   │   └── humanizer.js # Typing simulation & random delays
│   │
│   ├── utils/
│   │   ├── logger.js # Structured logging (Winston or Pino)
│   │   ├── sanitizer.js # Phone number normalization & input validation
│   │   └── errors.js # Custom error classes for error handling
│   │
│   ├── database/
│   │   ├── connection.js # SQLite connection pool
│   │   ├── schema.sql # Database schema DDL
│   │   └── queries.js # Prepared statements & query builders
│   │
│   └── index.js # Application entry point
│
├── data/ # CRITICAL: Must be mounted as Docker volume
│   ├── .wwebjs_auth/ # WhatsApp session (NEVER DELETE)
│   │   └── session-client/ # LocalAuth storage
│   ├── database.sqlite # Message history & metadata
│   └── logs/ # Rotating log files
│   ├── application.log
│   └── error.log
│
├── tests/
│   ├── unit/
```

```
│   ├── integration/
│   └── fixtures/
│
├── .env.example # Template for environment variables
├── .env # NEVER COMMIT - Actual secrets
├── .gitignore
├── .dockerignore
├── Dockerfile
├── docker-compose.yml # Local development environment
├── package.json
├── package-lock.json
└── README.md
```

3.2 Critical File: package.json

```
{
"name": "carl-bot",
"version": "2.0.0",
"type": "module",
"engines": {
"node": ">=18.0.0"
},
"scripts": {
"start": "node src/index.js",
"dev": "NODE_ENV=development node --watch src/index.js",
"test": "NODE_ENV=test node --test tests/**/*.test.js",
"docker:build": "docker build -t carl-bot:latest .",
"docker:run": "docker-compose up"
},
"dependencies": {
"whatsapp-web.js": "^1.23.0",
"puppeteer": "^21.0.0",
"openai": "^4.20.0",
"googleapis": "^126.0.0",
"node-telegram-bot-api": "^0.64.0",
"better-sqlite3": "^9.2.0",
"p-queue": "^7.4.0",
"winston": "^3.11.0",
"qrcode-terminal": "^0.12.0",
"dotenv": "^16.3.0"
},
"devDependencies": {
"@types/node": "^20.10.0"
}
}
```

---

4. Infrastructure & Deployment

4.1 The Chromium Problem (Critical Infrastructure Issue)

Problem: whatsapp-web.js requires a full Chromium browser installation. Official Node.js Docker images (node:18-alpine, node:18) do not include the 50+ system libraries required by Chromium.

Solution: Multi-stage Docker build with explicit dependency installation.

4.2 Production Dockerfile (Puppeteer-Ready)

Multi-stage build for optimized image size

FROM node:18-slim AS base

Install Chromium dependencies (Critical for Puppeteer)

These libraries are required for headless Chrome execution in Linux

RUN apt-get update && apt-get install -y
wget
gnupg
ca-certificates
procps
libxss1
libasound2
libatk-bridge2.0-0
libgtk-3-0
libgbm-dev
libnss3
libxcomposite1
libxdamage1
libxrandr2
libpango-1.0-0
libcairo2
libcups2
libdrm2
libatspi2.0-0
--no-install-recommends
&& rm -rf /var/lib/apt/lists/*

Set working directory

WORKDIR /usr/src/app

Install dependencies first (Docker layer caching optimization)

COPY package*.json ./
RUN npm ci --only=production

Copy application code

COPY . .

Configure Puppeteer to use system Chromium

ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD=true
ENV PUPPETEER_EXECUTABLE_PATH=/usr/bin/google-chrome-stable

Create data directory for volume mounting

RUN mkdir -p /usr/src/app/data

Run as non-root user (security best practice)

RUN useradd -m -u 1001 carlbot &&
chown -R carlbot:carlbot /usr/src/app
USER carlbot

Health check (optional but recommended)

HEALTHCHECK --interval=30s --timeout=10s --start-period=60s
CMD node -e "process.exit(0)"

Start application

CMD ["node", "src/index.js"]

4.3 Docker Compose for Local Development

version: '3.8'

services:
carl-bot:
build: .
container_name: carl-bot-dev
environment:
- NODE_ENV=development
- OPENAI_API_KEY=$OPENAI_APIKEY - TELEGRAM_BOTTOKEN =$
{TELEGRAM_BOT_TOKEN}
- TELEGRAM_ADMIN_ID=${TELEGRAM_ADMIN_ID}
volumes:
CRITICAL: Persistent volume for WhatsApp session
- ./data:/usr/src/app/data
Hot reload for development
- ./src:/usr/src/app/src
restart: unless-stopped
logging:
driver: "json-file"
options:
max-size: "10m"
max-file: "3"

4.4 Render.com Configuration

Service Type: Background Worker (not Web Service - no HTTP port exposure needed)

Critical Settings:

- **Build Command:** `npm ci --only=production`
- **Start Command:** `node src/index.js`
- **Environment:** Docker
- **Disk (MANDATORY for production):**
  - Mount Path: `/usr/src/app/data`
  - Size: 1GB
  - **Cost:** $0.25/GB/month (~$0.25/month)
  - **Consequence of omission:** WhatsApp QR code re-scan required after every deployment or container restart

Environment Variables (Render Dashboard):
NODE_ENV=production
OPENAI_API_KEY=sk-proj-xxxxxxxxxxxx
GOOGLE_SERVICE_ACCOUNT_JSON={"type":"service_account",...}
TELEGRAM_BOT_TOKEN=1234567890:ABCdefGHIjklMNOpqrsTUVwxyz
TELEGRAM_ADMIN_ID=123456789

WHATSAPP_SESSION_PATH=/usr/src/app/data/.wwebjs_auth
DATABASE_PATH=/usr/src/app/data/database.sqlite

Auto-Deploy: Connect GitHub repository, enable auto-deploy on `main` branch push.

---

5. Security Architecture

5.1 Authentication Mechanisms

A. WhatsApp Session Persistence (LocalAuth)

Challenge: whatsapp-web.js stores authentication session in a local directory. Container restarts destroy ephemeral filesystems.

Solution Implementation:

```
// src/services/whatsapp.js
import { Client, LocalAuth } from 'whatsapp-web.js';
import path from 'path';

const client = new Client({
authStrategy: new LocalAuth({
clientId: "carl-client",
dataPath: process.env.WHATSAPP_SESSION_PATH || './data/.wwebjs_auth'
}),
puppeteer: {
headless: true,
args: [
'--no-sandbox',
'--disable-setuid-sandbox',
'--disable-dev-shm-usage',
'--disable-accelerated-2d-canvas',
'--no-first-run',
'--no-zygote',
'--single-process',
'--disable-gpu'
]
}
});
```

Critical Note: The `dataPath` MUST point to a Docker volume mount. Without persistent storage, QR code re-authentication is required on every restart.

B. Google Calendar Service Account (Unattended Access)

Why Service Account vs OAuth2:

Aspect	OAuth2 User Flow	Service Account
Authentication	Requires browser redirect	JSON key file (static)
Token Refresh	Manual user interaction needed	Automatic (JWT signing)
Server Deployment	Complex state management	Zero-interaction
Use Case	User-facing applications	Server-to-server automation

Implementation Steps:

1. Create Service Account in Google Cloud Console
2. Download JSON key file (e.g., carl-bot-sa@project.iam.gserviceaccount.com)
3. **Critical Step:** Share your personal Google Calendar with the service account email
4. Grant permission: "Make changes to events"

```
// src/services/calendar.js
import { google } from 'googleapis';

const serviceAccount = JSON.parse(process.env.GOOGLE_SERVICE_ACCOUNT_JSON);

const auth = new google.auth.GoogleAuth({
credentials: serviceAccount,
scopes: ['https://www.googleapis.com/auth/calendar']
});

const calendar = google.calendar({ version: 'v3', auth });

export async function checkAvailability(startTime, endTime) {
const response = await calendar.freebusy.query({
requestBody: {
timeMin: startTime.toISOString(),
timeMax: endTime.toISOString(),
items: [{ id: 'primary' }]
}
});

return response.data.calendars.primary.busy;
}
```

5.2 Secrets Management

Never commit secrets. Use environment variables exclusively.

.env Structure:

OpenAI Configuration

OPENAI_API_KEY=sk-proj-xxxxxxxxxxxxxxxxxxxxxxxx
OPENAI_MODEL=gpt-4o
OPENAI_MAX_TOKENS=500

Google Cloud

GOOGLE_SERVICE_ACCOUNT_JSON='{"type":"service_account","project_id":"..."}'

Telegram Bot

TELEGRAM_BOT_TOKEN=1234567890:ABCdefGHIjklMNOpqrsTUVwxyz
TELEGRAM_ADMIN_ID=123456789

Application Config

NODE_ENV=production
LOG_LEVEL=info
DATABASE_PATH=/usr/src/app/data/database.sqlite
WHATSAPP_SESSION_PATH=/usr/src/app/data/.wwebjs_auth

.gitignore (Mandatory Entries):
.env
.env.local
data/
node_modules/
*.log

5.3 Input Sanitization & Validation

Threat Model: Malicious messages could attempt injection attacks or trigger unintended behavior.

Sanitization Strategy:

// src/utils/sanitizer.js

export function sanitizePhoneNumber(number) {
// Remove all non-digit characters except '+'
return number.replace(/[^\d+]/g, '');
}

export function sanitizeMessageContent(content) {
// Remove zero-width characters (potential hidden commands)
const cleaned = content.replace(/[\u200B-\u200D\uFEFF]/g, '');

// Truncate excessively long messages (DoS prevention)
const MAX_LENGTH = 4096;

```
return cleaned.slice(0, MAX_LENGTH);
}

export function validateUrl(url) {
try {
const parsed = new URL(url);
// Only allow https URLs
return parsed.protocol === 'https:';
} catch {
return false;
}
}
```

---

6. AI Strategy & Cost Optimization

6.1 Structured Output Schema (OpenAI JSON Mode)

Problem: LLMs can hallucinate invalid JSON or include unwanted conversational preambles.

Solution: Use OpenAI's Structured Output mode with explicit JSON Schema enforcement.

```
// src/ai/structuredOutput.js

export const MESSAGE_ANALYSIS_SCHEMA = {
type: "object",
properties: {
intent: {
type: "string",
enum: ["meeting_request", "information_query", "spam", "urgent", "casual_chat"]
},
urgency: {
type: "string",
enum: ["low", "medium", "high", "critical"]
},
category: {
type: "string",
enum: ["professional", "personal", "volunteer", "sports", "spam"]
},
requires_calendar: {
type: "boolean"
},
suggested_response: {
type: "string"
},
confidence: {
type: "number",
minimum: 0,
maximum: 1
}
},
```

required: ["intent", "urgency", "category", "requires_calendar", "suggested_response", "confidence"]
};

API Call Implementation:

```
// src/ai/openai.js
import OpenAI from 'openai';

const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

export async function analyzeMessage(message, conversationHistory) {
const response = await openai.chat.completions.create({
model: "gpt-4o",
messages: [
{
role: "system",
content: `You are C.A.R.L., the personal assistant of Jonas. You are efficient, concise, and professional. You have access to his calendar.

Your output MUST be a valid JSON object matching the provided schema. Never include explanations outside the JSON structure.`
},
...conversationHistory.slice(-3), // Last 3 messages for context
{
role: "user",
content: message.body
}
],
response_format: {
type: "json_schema",
json_schema: {
name: "message_analysis",
schema: MESSAGE_ANALYSIS_SCHEMA
}
},
max_tokens: 300,
temperature: 0.3
});

return JSON.parse(response.choices[0].message.content);
}
```

6.2 Cost Optimization Strategy

OpenAI Pricing (November 2025):

- GPT-4o: $5.00 / 1M input tokens, $15.00 / 1M output tokens
- GPT-4o-mini: $0.15 / 1M input tokens, $0.60 / 1M output tokens

Optimization Techniques:

1. **Context Window Management:** Only send last 3 messages + system prompt (not entire conversation history)
2. **Model Selection:** Use GPT-4o-mini for routine classification, GPT-4o for complex decision-making
3. **Token Limiting:** Set `max_tokens: 300` for responses (conversational responses rarely need more)
4. **Caching Strategy:** Cache common intents (e.g., "meeting request" patterns) for 5 minutes to avoid duplicate API calls

Estimated Monthly Cost (100 messages/day):

- Input: ~3,000 messages × 150 tokens = 450k tokens = $0.07
- Output: ~3,000 responses × 100 tokens = 300k tokens = $4.50
- **Total: ~$5/month**

6.3 System Prompt Engineering

// src/ai/prompts.js

export const SYSTEM_PROMPT = `You are C.A.R.L. (Communication Assistant for Routing & Logistics), the personal executive assistant of Jonas.

Personality Traits:

- Efficient and concise (no unnecessary verbosity)
- Professional yet approachable
- Proactive problem-solver
- Calendar-aware and time-conscious

Capabilities:

1. Access to Jonas's Google Calendar (read/write)
2. Message classification (Professional/Personal/Spam/Urgent)
3. Intent extraction (meeting requests, information queries, casual chat)
4. Response drafting in Jonas's communication style

Response Guidelines:

- Always respond in French (Jonas's primary language)
- Use formal tone for professional contacts, casual for personal
- For meeting requests: Check calendar and propose 3 available slots
- For urgent matters: Notify admin via Telegram
- For spam: Politely decline or ignore

Output Format:

Your response MUST be a JSON object matching the provided schema. Never include conversational text outside the JSON structure.`;

export function buildContextPrompt(conversationHistory) {
if (conversationHistory.length === 0) return '';

```
const summary = conversationHistory.slice(-5).map(msg =>
${msg.fromMe ? 'Jonas' : 'Contact'}: ${msg.body}
).join('\n');

return \n\n## Recent Conversation Context:\n${summary};
}
```

---

7. Anti-Ban Strategy (WhatsApp TOS Compliance)

7.1 Risk Assessment

WhatsApp Terms of Service Position:
Using automated clients on personal accounts violates Section 4.5 of WhatsApp's Terms of Service. However, enforcement is primarily targeted at:

- Bulk messaging / spam operations
- Commercial marketing automation
- High-frequency API abuse

Risk Mitigation for Personal Use:

- Single-user assistant (not multi-tenant)
- Human-like interaction patterns
- Rate limiting enforcement
- No broadcast messaging

7.2 Rate Limiting Implementation

```
// src/middleware/humanizer.js

export class HumanBehaviorSimulator {
constructor() {
this.lastMessageTime = new Map(); // sender -> timestamp
this.messageCount = new Map(); // sender -> count in last minute
}

async simulateTyping(chat, messageLength) {
// Calculate realistic typing time based on message length
// Average human typing speed: 40 words/minute = ~200 characters/minute
const baseDelay = (messageLength / 200) * 60 * 1000; // milliseconds

  // Add randomness (+/- 30%)
  const randomFactor = 0.7 + (Math.random() * 0.6);
  const typingDuration = Math.min(baseDelay * randomFactor, 5000); // Max 5 sec

  await chat.sendStateTyping();
  await this.sleep(typingDuration);

}
```

```
async enforceRateLimit(senderId) {
const now = Date.now();
const lastSent = this.lastMessageTime.get(senderId) || 0;
const timeSinceLastMessage = now - lastSent;

  // Minimum 2 seconds between messages to same sender
  if (timeSinceLastMessage < 2000) {
    await this.sleep(2000 - timeSinceLastMessage);
  }

  // Update tracking
  this.lastMessageTime.set(senderId, Date.now());

  // Check messages per minute
  const countKey = `${senderId}-${Math.floor(now / 60000)}`;
  const count = this.messageCount.get(countKey) || 0;

  if (count >= 5) {
    throw new Error('Rate limit exceeded: max 5 messages per minute per sender
  }

  this.messageCount.set(countKey, count + 1);

  // Cleanup old entries
  this.cleanupOldEntries();

}

sleep(ms) {
return new Promise(resolve => setTimeout(resolve, ms));
}

cleanupOldEntries() {
const now = Date.now();
for (const [key, timestamp] of this.lastMessageTime.entries()) {
if (now - timestamp > 3600000) { // 1 hour
this.lastMessageTime.delete(key);
}
}
}
}
```

7.3 Detection Avoidance Patterns

- **Never respond in < 1 second** - Use random delays between 2-5 seconds
- **Typing indicator** - Always show "typing..." state before sending
- **Read receipts** - Enable read receipt simulation with delays
- **Online status** - Maintain realistic online/offline patterns
- **Message variance** - Add slight variations to template responses
- **No broadcast messaging** - Never send identical message to > 5 recipients

---

8. Database Schema & Data Persistence

8.1 SQLite Schema

```
-- schema.sql

CREATE TABLE IF NOT EXISTS conversations (
id INTEGER PRIMARY KEY AUTOINCREMENT,
chat_id TEXT NOT NULL,
sender_id TEXT NOT NULL,
message_id TEXT UNIQUE NOT NULL,
body TEXT NOT NULL,
timestamp INTEGER NOT NULL,
is_from_me BOOLEAN NOT NULL DEFAULT 0,
media_type TEXT,
has_media BOOLEAN DEFAULT 0,
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
INDEX idx_chat_timestamp (chat_id, timestamp),
INDEX idx_sender_timestamp (sender_id, timestamp)
);

CREATE TABLE IF NOT EXISTS message_metadata (
id INTEGER PRIMARY KEY AUTOINCREMENT,
message_id TEXT NOT NULL,
intent TEXT,
urgency TEXT,
category TEXT,
confidence REAL,
ai_analysis JSON,
processed_at DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (message_id) REFERENCES conversations(message_id)
);

CREATE TABLE IF NOT EXISTS contacts (
id INTEGER PRIMARY KEY AUTOINCREMENT,
contact_id TEXT UNIQUE NOT NULL,
name TEXT,
phone_number TEXT,
is_whitelisted BOOLEAN DEFAULT 0,
is_blacklisted BOOLEAN DEFAULT 0,
category TEXT,
notes TEXT,
```

```
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS daily_reports (
id INTEGER PRIMARY KEY AUTOINCREMENT,
report_date DATE NOT NULL,
total_messages INTEGER,
urgent_count INTEGER,
professional_count INTEGER,
personal_count INTEGER,
spam_count INTEGER,
summary TEXT,
generated_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS calendar_events (
id INTEGER PRIMARY KEY AUTOINCREMENT,
event_id TEXT UNIQUE NOT NULL,
summary TEXT NOT NULL,
start_time DATETIME NOT NULL,
end_time DATETIME NOT NULL,
location TEXT,
related_message_id TEXT,
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (related_message_id) REFERENCES conversations(message_id)
);
```

8.2 Database Connection Manager

```
// src/database/connection.js
import Database from 'better-sqlite3';
import fs from 'fs';
import path from 'path';

class DatabaseManager {
constructor(dbPath) {
this.dbPath = dbPath;
this.db = null;
}

initialize() {
// Ensure data directory exists
const dir = path.dirname(this.dbPath);
if (!fs.existsSync(dir)) {
fs.mkdirSync(dir, { recursive: true });
}

this.db = new Database(this.dbPath);
this.db.pragma('journal_mode = WAL'); // Write-Ahead Logging for performanc
```

```
    this.db.pragma('foreign_keys = ON');

    // Load schema
    const schema = fs.readFileSync('./src/database/schema.sql', 'utf8');
    this.db.exec(schema);
```

}

insertMessage(chatId, senderId, messageId, body, timestamp, isFromMe) {
const stmt = this.db.prepare(INSERT INTO conversations (chat_id, sender_id, message_id, body, timestamp, is_from_me) VALUES (?, ?, ?, ?, ?, ?));
return stmt.run(chatId, senderId, messageId, body, timestamp, isFromMe ? 1 : 0);
}

getRecentMessages(chatId, limit = 10) {
const stmt = this.db.prepare(SELECT * FROM conversations WHERE chat_id = ? ORDER BY timestamp DESC LIMIT ?);
return stmt.all(chatId, limit);
}

close() {
if (this.db) {
this.db.close();
}
}
}

export default DatabaseManager;

---

9. Message Processing Pipeline (Core Logic)

9.1 Main Event Handler

// src/services/whatsapp.js
import { Client, LocalAuth } from 'whatsapp-web.js';
import { analyzeMessage } from '../ai/openai.js';
import { HumanBehaviorSimulator } from '../middleware/humanizer.js';
import { checkAvailability } from './calendar.js';
import DatabaseManager from '../database/connection.js';
import PQueue from 'p-queue';

const humanizer = new HumanBehaviorSimulator();
const db = new DatabaseManager(process.env.DATABASE_PATH);
const queue = new PQueue({ concurrency: 3 }); // Max 3 simultaneous processings

export async function initializeWhatsAppClient() {
const client = new Client({
authStrategy: new LocalAuth({
clientId: "carl-client",

```
dataPath: process.env.WHATSAPP_SESSION_PATH
}),
puppeteer: {
headless: true,
args: [
'--no-sandbox',
'--disable-setuid-sandbox',
'--disable-dev-shm-usage'
]
}
});

client.on('qr', (qr) => {
console.log('QR RECEIVED', qr);
// In production: Send QR to Telegram admin bot
});

client.on('ready', () => {
console.log('C.A.R.L. is ready');
});

client.on('message_create', async (message) => {
// Add to queue to prevent concurrent processing
queue.add(() => handleMessage(client, message));
});

await client.initialize();
return client;
}

async function handleMessage(client, message) {
try {
// Gatekeeper: Skip own messages
if (message.fromMe) return;
```

```
// Gatekeeper: Skip group messages (optional)
const chat = await message.getChat();
if (chat.isGroup) return;

// Gatekeeper: Skip status updates
if (message.isStatus) return;

// Store message in database
db.insertMessage(
  chat.id._serialized,
  message.from,
  message.id._serialized,
```

```
    message.body,
    message.timestamp,
    false
  );

  // Retrieve conversation history
  const history = db.getRecentMessages(chat.id._serialized, 5);

  // AI Analysis
  const analysis = await analyzeMessage(message, history);

  // Execute action based on intent
  if (analysis.intent === 'meeting_request' && analysis.requires_calendar) {
    const availability = await checkAvailability(new Date(), new Date(Date.now() +
    // Process availability and update suggested_response
  }

  // Simulate human behavior
  await humanizer.enforceRateLimit(message.from);
  await humanizer.simulateTyping(chat, analysis.suggested_response.length);

  // Send response
  await chat.sendMessage(analysis.suggested_response);
```

```
} catch (error) {
console.error('Error handling message:', error);
// Send error notification to Telegram admin
}
}
```

---

10. Deployment Checklist

10.1 Pre-Deployment Requirements

1. **OpenAI Account**
   - API key generated (`sk-proj-...`)
   - Minimum $5 credit added (API requires prepaid balance)
   - Rate limits reviewed (Tier 1: 500 RPM / 200k TPM)
2. **Google Cloud Setup**
   - Project created in Google Cloud Console
   - Calendar API enabled (APIs & Services → Library)
   - Service Account created (IAM → Service Accounts)

- JSON key downloaded
- Personal Google Calendar shared with service account email

3. **Telegram Bot**
   - Bot created via @BotFather
   - Token retrieved
   - Personal Telegram ID obtained (via @userinfobot)
4. **Development Environment**
   - Node.js v18+ installed
   - Docker Desktop installed (for local testing)
   - Git repository initialized

10.2 Render.com Deployment Steps

1. **Create Render Account** and connect GitHub repository
2. **Create Background Worker Service:**
   - Name: `carl-bot-production`
   - Environment: Docker
   - Branch: `main`
   - Build Command: `docker build -t carl-bot .`
   - Start Command: Defined in Dockerfile (`CMD ["node", "src/index.js"]`)
3. **Add Environment Variables** (see Section 5.2)
4. **Create Disk Volume:**
   - Name: `carl-bot-data`
   - Mount Path: `/usr/src/app/data`
   - Size: 1GB
5. **Deploy** and monitor logs for QR code
6. **Initial Authentication:**
   - Retrieve QR code from logs or Telegram notification
   - Scan with WhatsApp mobile app
   - Wait for "C.A.R.L. is ready" confirmation

10.3 Post-Deployment Monitoring

Key Metrics to Monitor:

- Container restarts (should be 0 after successful QR scan)
- Message processing latency (target: <3 seconds)
- OpenAI API errors (rate limits, token exhaustion)
- Database size growth (estimate: 1MB per 1000 messages)
- Memory usage (target: <512MB)

Log Monitoring Commands:

```
# View real-time logs
render logs -f carl-bot-production
```

Check container status

render ps carl-bot-production

View disk usage

render disk usage carl-bot-data

---

11. Operational Procedures

11.1 QR Code Re-Authentication

Trigger Conditions:

- WhatsApp session expired (typically after 14 days of inactivity)
- Volume corruption or data loss
- Manual logout from WhatsApp mobile app

Recovery Procedure:

1. Check Render logs for QR code ASCII art
2. Alternatively, implement Telegram notification:

```
// In whatsapp.js
client.on('qr', async (qr) => {
console.log('QR RECEIVED', qr);

// Generate QR code image
const qrImage = await qrcode.toDataURL(qr);

// Send to Telegram admin
await telegramBot.sendPhoto(
process.env.TELEGRAM_ADMIN_ID,
Buffer.from(qrImage.split(',')[1], 'base64'),
{ caption: '🤖 C.A.R.L. requires re-authentication. Scan this QR code.' }
);
});
```

3. Scan QR code with WhatsApp mobile app (Settings → Linked Devices → Link a Device)
4. Wait for "ready" event confirmation

11.2 Emergency Shutdown

Command via Telegram Admin Bot:
/stop - Graceful shutdown (finish processing current messages)
/force_stop - Immediate termination

Implementation:

```
// src/services/telegram.js
import TelegramBot from 'node-telegram-bot-api';
```

```
const bot = new TelegramBot(process.env.TELEGRAM_BOT_TOKEN, { polling: true });

bot.onText(//stop/, async (msg) => {
if (msg.from.id.toString() !== process.env.TELEGRAM_ADMIN_ID) return;

await bot.sendMessage(msg.chat.id, '🛑 Initiating graceful shutdown...');

// Wait for queue to drain
await queue.onIdle();

// Close database connection
db.close();

// Disconnect WhatsApp client
await whatsappClient.destroy();

process.exit(0);
});
```

11.3 Daily Briefing Generation

Scheduled Task (Cron-like implementation):

```
// src/services/reports.js
import { CronJob } from 'cron';

export function scheduleDailyBriefing() {
// Every day at 8:00 AM
const job = new CronJob('0 8 * * *', async () => {
const yesterday = new Date(Date.now() - 86400000);
const report = await generateDailyReport(yesterday);

  await sendToTelegram(report);

});

job.start();
}

async function generateDailyReport(date) {
const stats = db.prepare( SELECT COUNT(*) as total, SUM(CASE WHEN urgency = 'high' OR urgency = 'critical' THEN 1 ELSE 0 END) as urgent, SUM(CASE WHEN category = 'professional' THEN 1 ELSE 0 END) as professional, SUM(CASE WHEN category = 'personal' THEN 1 ELSE 0 END) as personal, SUM(CASE WHEN category = 'spam' THEN 1 ELSE 0 END) as spam FROM conversations WHERE DATE(timestamp, 'unixepoch') = DATE(?)
).get(date.toISOString().split('T')[0]);

return {
date,
...stats,
summary: 📊 Daily Report for ${date.toLocaleDateString()}\n\n +
Total Messages: ${stats.total}\n +
```

```
Urgent: ${stats.urgent}\n +
Professional: ${stats.professional}\n +
Personal: ${stats.personal}\n +
Spam: ${stats.spam}
};
}
```

---

12. Testing Strategy

12.1 Unit Testing

```
// tests/unit/sanitizer.test.js
import { describe, it } from 'node:test';
import assert from 'node:assert';
import { sanitizePhoneNumber, sanitizeMessageContent } from '../../src/utils/sanitizer.js';

describe('Sanitizer', () => {
it('should remove non-digit characters from phone numbers', () => {
const input = '+33 (0)6 12 34 56 78';
const expected = '+33612345678';
assert.strictEqual(sanitizePhoneNumber(input), expected);
});

it('should remove zero-width characters from messages', () => {
const input = 'Hello\u200BWorld';
const expected = 'HelloWorld';
assert.strictEqual(sanitizeMessageContent(input), expected);
});

it('should truncate excessively long messages', () => {
const input = 'A'.repeat(5000);
const output = sanitizeMessageContent(input);
assert.strictEqual(output.length, 4096);
});
});
```

12.2 Integration Testing

Mock WhatsApp Client for CI/CD:

```
// tests/integration/message-pipeline.test.js
import { describe, it, before, after } from 'node:test';
import assert from 'node:assert';
import { createMockClient } from './mocks/whatsapp-mock.js';

describe('Message Pipeline', () => {
let mockClient;

before(async () => {
mockClient = await createMockClient();
});
```

```
it('should process meeting request and check calendar', async () => {
const testMessage = {
from: 'test@s.whatsapp.net',
body: 'On peut se voir mardi prochain ?',
timestamp: Date.now()
};

  const result = await handleMessage(mockClient, testMessage);

  assert.strictEqual(result.intent, 'meeting_request');
  assert.strictEqual(result.requires_calendar, true);

});

after(() => {
mockClient.destroy();
});
});
```

---

13. Roadmap & Future Enhancements

Phase 1: MVP (Weeks 1-4)

- [ ] Core WhatsApp connectivity
- [ ] OpenAI integration with structured output
- [ ] Basic message classification
- [ ] SQLite persistence
- [ ] Docker deployment on Render

Phase 2: Intelligence (Weeks 5-8)

- [ ] Google Calendar integration
- [ ] Meeting request detection and slot proposal
- [ ] Conversation context enrichment
- [ ] Telegram admin bot notifications

Phase 3: Personalization (Weeks 9-12)

- [ ] Writing style learning (fine-tuning or few-shot prompting)
- [ ] Contact categorization (whitelist/blacklist)
- [ ] Automated response templates
- [ ] Daily briefing generation

Phase 4: Advanced Features (Future)

- [ ] Multi-language support
- [ ] Voice message transcription (Whisper API)
- [ ] Image analysis (GPT-4 Vision)
- [ ] Integration with Notion/Slack
- [ ] Machine learning-based spam detection (local model)

---

14. Cost Analysis

14.1 Monthly Operational Costs

Service	Cost (USD)	Notes
Render.com (Starter)	$7.00	Background Worker with 512MB RAM
Render Disk (1GB)	$0.25	Persistent volume for WhatsApp session
OpenAI API	$5.00	Estimated for 100 messages/day
Google Cloud (Calendar API)	$0.00	Free tier (sufficient for personal use)
Telegram Bot API	$0.00	Completely free
Total	**$12.25/month**	

Table 2: Monthly operational cost breakdown

14.2 Cost Optimization Opportunities

- **Use GPT-4o-mini** for routine classification (90% cost reduction for those calls)
- **Implement local caching** for common intents (Redis or in-memory)
- **Batch calendar queries** to reduce API calls
- **Use Render free tier** initially (75 hours/month, sufficient for testing)

---

15. Legal & Compliance

15.1 WhatsApp Terms of Service

Status: Gray area. Automation of personal accounts violates ToS Section 4.5.

Mitigation Strategy:

- Personal use only (not commercial)
- Rate limiting to mimic human behavior
- No spam or broadcast messaging
- Transparent operation (bot identifies itself if questioned)

Disclaimer for Users:

> This software is provided for educational and personal productivity purposes. Use at your own risk. The developers assume no liability for account bans or ToS violations.

15.2 Data Privacy (GDPR Compliance)

Data Minimization:

- Only store message content necessary for context (last 10 messages)
- No sensitive data exported or shared with third parties (except OpenAI for processing)
- Database encrypted at rest (SQLite with SQLCipher if needed)

User Rights:

- Right to access: Database query scripts provided
- Right to deletion: `DELETE FROM conversations WHERE chat_id = ?`
- Right to portability: Export function to JSON

---

16. Troubleshooting Guide

16.1 Common Issues

Issue: "QR code expired" error
Cause: QR code timeout (60 seconds)
Solution: Restart container, retrieve new QR code immediately

Issue: "Cannot find module 'puppeteer'"
Cause: Incomplete npm install
Solution: Run `npm ci` instead of `npm install`, ensure `package-lock.json` exists

Issue: "Session closed" after container restart
Cause: Volume not mounted correctly
Solution: Verify Docker volume mount path matches `WHATSAPP_SESSION_PATH` environment variable

Issue: OpenAI API error "Insufficient quota"
Cause: No prepaid credit in OpenAI account
Solution: Add minimum $5 credit in OpenAI dashboard → Billing

Issue: Google Calendar permission denied
Cause: Calendar not shared with service account
Solution: Google Calendar → Settings → Share with [service-account-email], grant "Make changes to events"

16.2 Debug Mode

Enable verbose logging:

```
// src/utils/logger.js
import winston from 'winston';
```

```
const logger = winston.createLogger({
level: process.env.LOG_LEVEL || 'info',
format: winston.format.combine(
winston.format.timestamp(),
winston.format.json()
),
transports: [
new winston.transports.File({ filename: 'data/logs/error.log', level: 'error' }),
new winston.transports.File({ filename: 'data/logs/application.log' }),
new winston.transports.Console({
format: winston.format.simple()
})
]
});

export default logger;
```

Set `LOG_LEVEL=debug` in environment variables for detailed output.

---

17. Conclusion

C.A.R.L. represents a production-grade personal assistant system that balances technical complexity with operational simplicity. The architecture prioritizes:

1. **Resilience:** Queue-based message processing prevents overload
2. **Persistence:** Docker volumes ensure session continuity across deployments
3. **Security:** Service accounts and environment variables protect credentials
4. **Compliance:** Rate limiting and human behavior simulation reduce ban risk
5. **Cost-Efficiency:** Optimized AI usage and free-tier services minimize operational costs

Critical Success Factors:

- Persistent storage configuration (Render Disk mount)
- Proper Chromium dependencies in Docker image
- OpenAI Structured Output implementation
- Service Account calendar sharing
- Rate limiting enforcement

Next Steps:

1. Initialize Git repository with provided structure
2. Configure all API credentials (OpenAI, Google, Telegram)
3. Test locally with Docker Compose
4. Deploy to Render with persistent disk
5. Scan QR code and validate message processing
6. Monitor logs and iterate on prompts

This document serves as the single source of truth for C.A.R.L. implementation. All code examples are production-ready and require minimal modification for deployment.

Appendix A: Complete Environment Variables Reference

Application

```
NODE_ENV=production
LOG_LEVEL=info
PORT=3000
```

WhatsApp

```
WHATSAPP_SESSION_PATH=/usr/src/app/data/.wwebjs_auth
```

OpenAI

```
OPENAI_API_KEY=sk-proj-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
OPENAI_MODEL=gpt-4o
OPENAI_MAX_TOKENS=500
OPENAI_TEMPERATURE=0.3
```

Google Calendar

```
GOOGLE_SERVICE_ACCOUNT_JSON={"type":"service_account","project_id":"..."}
GOOGLE_CALENDAR_ID=primary
```

Telegram

```
TELEGRAM_BOT_TOKEN=1234567890:ABCdefGHIjklMNOpqrsTUVwxyz
TELEGRAM_ADMIN_ID=123456789
```

Database

```
DATABASE_PATH=/usr/src/app/data/database.sqlite
```

Rate Limiting

```
MAX_MESSAGES_PER_MINUTE=5
MIN_DELAY_BETWEEN_MESSAGES=2000
MAX_TYPING_DELAY=5000
```

Features

ENABLE_DAILY_BRIEFING=true
DAILY_BRIEFING_TIME=08:00
ENABLE_AUTO_RESPONSE=true
ENABLE_CALENDAR_INTEGRATION=true

Appendix B: Docker Compose Full Configuration

version: '3.8'

services:
carl-bot:
build:
context: .
dockerfile: Dockerfile
container_name: carl-bot
environment:
- NODE_ENV=$NODE_ENV : -production - OPENAI_API_KEY =${OPENAI_API_KEY}
- OPENAI_MODEL=$OPENAI_MODEL : -gpt - 4o - TELEGRAM_BOT_TOKEN =
{TELEGRAM_BOT_TOKEN}
- TELEGRAM_ADMIN_ID=
$TELEGRAM_ADMIN_ID - GOOGLE_SERVICE_ACCOUNT_JSON =
{GOOGLE_SERVICE_ACCOUNT_JSON}
- WHATSAPP_SESSION_PATH=/usr/src/app/data/.wwebjs_auth
- DATABASE_PATH=/usr/src/app/data/database.sqlite
- LOG_LEVEL=${LOG_LEVEL:-info}
volumes:
- carl-data:/usr/src/app/data
- ./src:/usr/src/app/src:ro # Development hot-reload
restart: unless-stopped
logging:
driver: "json-file"
options:
max-size: "10m"
max-file: "3"
healthcheck:
test: ["CMD", "node", "-e", "process.exit(0)"]
interval: 30s
timeout: 10s
retries: 3
start_period: 60s

volumes:
carl-data:
driver: local

Appendix C: References

[1] WhatsApp Web.js Documentation. (2025). Retrieved from https://wwebjs.dev/

[2] OpenAI API Reference. (2025). Structured Outputs. Retrieved from https://platform.openai.com/docs/guides/structured-outputs

[3] Google Calendar API Documentation. (2025). Service Accounts. Retrieved from https://developers.google.com/calendar/api/guides/service-accounts

[4] Puppeteer Documentation. (2025). Running in Docker. Retrieved from https://pptr.dev/guides/docker

[5] Render Documentation. (2025). Persistent Disks. Retrieved from https://render.com/docs/disks