Document 01 – Bouw van de Gemeente Chatbot

1. Doel en scope
- **Probleemstelling:** burgers willen 24/7 antwoord op vragen rond klachten, burgerzaken en servicekanalen.
- **Oplossing:** een privacy-veilige conversational flow die in het Nederlands antwoordt, routes aanbiedt (bijv. klacht indienen) en via API met het bestaande Laravel-platform praat.
- **Scope van dit document:** uitlegen hoe de chatbot is opgebouwd, welke codeonderdelen samenwerken en welke configuratie nodig is om het geheel draaiend te houden.

2. Architectuur in lagen

2.1 Backend service – `app/Services/ChatbotService.php`
- **Intent-detectie:** `setupIntents()` definieert sleutelwoorden voor ~20 intenten (klachten, burgerzaken, verkeer, afval, belastingen, noodsituaties). De `detectIntent()`-methode loopt deze patronen lineair af en retourneert de eerste match.
- **Kennisbank:** `loadKnowledgeBase()` hydrateert statische datasets (statusbeschrijvingen, contactgegevens, dienst-specifieke openingstijden, tariefinformatie) die door de response-methodes worden gebruikt.
- **Conversielogica:** `processMessage()` normaliseert de input, bepaalt de intent en routeert naar één van de ~25 specifieke `get...()`-responsehelpers. Elke helper levert een gestructureerde array met `type`, `message`, optionele `quick_replies` en `action_button`.
- **Logging:** elke call logt naar het `privacy_safe` kanaal met intent, berichtlengte en response-type zodat analytics draait zonder PII (zie `Log::channel('privacy_safe')->info(...)`).

2.2 API-laag – `app/Http/Controllers/Api/ChatController.php`
- **Validatie:** `chat()` eist `message` (max. 500 tekens) en accepteert optioneel `session_id`.
- **Afhandeling:** na `ChatbotService::processMessage()` wordt de reactie teruggestuurd als JSON `{ success, response, session_id, timestamp }`.
- **Beveiliging/logging:** fouten worden afgevangen en leiden tot beveiligde melding + fallback-antwoord; `PrivacyLogger::logUserAction()` registreert elk gesprek en verhasht sessiegegevens.
- **Overige endpoints:** `welcome()` en `faq()` leveren statische content voor het initialiseren van de widget.

2.3 Routes en rate limiting – `routes/api.php` en `app/Providers/AppServiceProvider.php`
- `/api/chat` routes leven in een `throttle:api` groep en hebben een extra limiter `RateLimiter::for('chat', ...)->perMinute(20)` gebaseerd op IP-adres; zo kan de chatbot los worden begrensd van de algemene API (60 req/min).

2.4 Frontend widget – `resources/js/chatbot.js`
- **Widget lifecycle:** `GemeenteChatbot` class maakt floating button + venster aan, injecteert CSS en blijft buiten de bestaande layout.
- **Sessies:** een pseudo-ID (`chat_<timestamp>_<random>`) wordt bij het opstarten aangemaakt en meegestuurd.
- **UX features:** quick replies, typing-indicator, karakters teller, action buttons, smooth animaties en responsive gedrag (CSS media queries in `getChatStyles()`).
- **API-koppeling:** `loadWelcomeMessage()` haalt `/api/chat/welcome`, `sendMessage()` post naar `/api/chat` en verwerkt `quick_replies` en `action_button`.

2.5 Privacy en auditing – `app/Services/PrivacyLogger.php` + `config/logging.php`
- `privacy_safe`, `security` en `audit` kanalen zijn gedefinieerd als daily logs met eigen retentie (30/90/365 dagen).

- `PrivacyLogger` verwijdert velden die PII kunnen bevatten (`isPotentialPII()`), hashed IP/user agent (`hashIp`, `hashUserAgent`) en biedt helper-methodes voor user actions, security events en audits.

3. Conversiestroom end-to-end
1. **Widget initieert** `/api/chat/welcome` en toont standaardknoppen.
2. **Gebruiker verzendt** tekst → frontend stuurt `{ message, session_id }` naar `/api/chat`.
3. **Validatie** in controller; bij succes roept deze `ChatbotService::processMessage()`.
4. **Intent match** → response helper voegt tekst, snelle opties en eventueel `action_button`.
5. **Logging**: service logt op `privacy_safe`, controller logt via `PrivacyLogger`.
6. **JSON-response** terug naar widget → UI rendert botbericht + nieuwe quick replies.
7. **Foutpad**: exception resulteert in fallback-antwoord + melding `Er is een fout opgetreden...`.

4. Functionele dekking
- **Klachten:** statusuitleg, ID-hulp, nieuwe klacht indienen inclusief stappenplan en link (`/klacht/status`, `/complaint/create`).
- **Burgerzaken:** paspoort, uittreksel, verhuizen, trouwen – elk met benoemde documenten, kosten en afspraken.
- **Contact/service:** telefoonnr, email, openingstijden, locaties (gemeentehuis, milieupark).
- **Verkeer & parkeren:** tarieven, vergunningen, blauwe zone, verkeerstoestemmingen.
- **Afval & milieu:** inzameldagen, grofvuilkosten, milieustraat info.
- **Belastingen & sociale zaken:** OZB, riool, kwijtschelding, bijstand, schuldhulp.
- **Events & digitale diensten:** seizoensevents, MijnGemeente login, app-informatie.
- **Noodsituaties:** directe instructies voor calamiteiten.

5. Data en herbruikbaarheid
- Alle vaste teksten en cijfers komen uit `knowledgeBase` zodat aanpassingen slechts op één plaats nodig zijn.
- Intents zijn eenvoudige substring-matches; uitbreiden = nieuwe entry in `$intents` en corresponderende `get...()` methode.
- Responses gebruiken uniforme structuur (`type`, `message`, `quick_replies`, optioneel `action_button`), wat front-end parsing versimpelt.

6. Deploy & beheer
- **Build pipeline:** `npm run build` genereert Vite assets, `php artisan migrate` beheert database, `php artisan config:cache` versnelt productie.
- **Monitoring:** controleer `storage/logs/application.log` (privacy_safe) voor gebruikscijfers en `storage/logs/security.log` voor errors.
- **Configuratie toggles:** rate limits via `AppServiceProvider`, logging via `.env` (`LOG_LEVEL`) en API-prefix via `RouteServiceProvider`.
- **Uitbreidingstips:** maak voor nieuwe intenten eerst knowledge-entry → voeg keywords → schrijf `getXInfo()` → test via `php artisan test` of Postman.

Met bovenstaande componenten vormt de chatbot een gesloten keten van UX, API, businesslogica en privacy-by-design logging die direct inzetbaar is op de gemeente website.