Document 03 – Uitgebreide Uitleg van de Belangrijkste Code

Deze gids zoomt in op de kernbestanden die de chatbot, de API en de belangrijkste datamodellen aandrijven. Elk onderdeel beschrijft de *waarom* achter de code zodat uitbreidingen doelgericht kunnen gebeuren.

1. Chatbot businesslogica – `app/Services/ChatbotService.php`

1.1 Kernstate en constructor
- `private array $knowledgeBase` bevat alle vaste teksten, cijfers en adressen.
- `private array $intents` definieert de keywordlijsten per intent.
- In `__construct()` wordt eerst `loadKnowledgeBase()` (datasets) en daarna `setupIntents()` (keywordmapping) aangeroepen zodat elke request over dezelfde in-memory structuur beschikt.

1.2 Verwerkingspijplijn
```php
public function processMessage(string $message): array
{
    $message = $this->normalizeMessage($message);
    $intent = $this->detectIntent($message);
    $response = $this->generateResponse($intent, $message);

    Log::channel('privacy_safe')->info('Chatbot interaction', [
        'intent' => $intent,
        'message_length' => strlen($message),
        'response_type' => $response['type'] ?? 'text',
    ]);

    return $response;
}
```

1. **Normaliseren:** `normalizeMessage()` lowercaset en trimt zodat `setupIntents()` nooit hoofdlettergevoelig hoeft te zijn.
2. **Intent bepalen:** `detectIntent()` loopt `foreach` door alle intent arrays en gebruikt `str_contains`. Zodra er een match is wordt direct teruggekeerd om performance voorspelbaar te houden.
3. **Response genereren:** `generateResponse()` routeert via `switch` naar een `getXInfo()` helper.
4. **Logging:** alleen afgeleide metadata (lengte, intent, response-type) wordt gelogd; het daadwerkelijke bericht wordt nooit opgeslagen voor privacy-redenen.

1.3 Belangrijkste helpers
- **Basisinteractie:** `getGreeting()`, `getThankYou()` leveren vriendelijke openings-/slotberichten met quick replies.
- **Klachtenstroom:** `getStatusInfo()`, `getKlachtIdHelp()`, `getComplaintSubmissionInfo()` lezen respectievelijk statusbeschrijvingen, ID-instructies en stappenplan uit de kennisbank en voegen CTA's zoals `Status Opzoeken` toe.
- **Burgerzaken & diensten:** `getPassportInfo()`, `getMovingInfo()`, `getTaxInfo()` bevatten specifieke teksten en verwijzingen naar tarieven of documenten. Hierdoor hoeft de frontend geen kennis te hebben van gemeentelijke processen.
- **Noodgevallen:** `getEmergencyInfo()` geeft directe instructies; deze categorie staat bewust bovenaan in `generateResponse()` zodat ze niet overschaduwd wordt door generieke intenten.

1.4 Knowledge base structuur
```php
$this->knowledgeBase = [
    'status_info' => [...],
    'contact' => ['phone' => '14 020', ...],
    'services' => [
        'burgerzaken' => ['hours' => '08:30-17:00', ...],
```

```
        'waste' => ['household' => 'Dinsdag', ...],
    ],
    'locations' => [...],
    'events' => [...],
];
```

- De dataset is modulair: nieuwe groepen kunnen worden toegevoegd zonder `processMessage()` te wijzigen.
- Door waarden zoals `contact.phone` te hergebruiken in meerdere responses (bedankt-bericht, klacht-ID hulp) blijft informatie consistent.

2. API-controller – `app/Http/Controllers/Api/ChatController.php`

2.1 POST `/api/chat`
1. **Validatie:** `message` verplicht, max 500 tekens; `session_id` optioneel maar wordt ingekort tot 100 tekens ter bescherming.
2. **Sessiebeheer:** `session()->getId()` fungeert als fallback zodat er altijd een ID aanwezig is voor logging en rate limiting.
3. **Servicecall:** `$this->chatbotService->processMessage($message)` levert de payload voor de frontend.
4. **Privacy logging:** `PrivacyLogger::logUserAction('chatbot_interaction', [...])` hashed de sessie (`hash('sha256', $sessionId)`) zodat gedrag inzichtelijk is zonder directe identifiers.
5. **Response:** consistente structuur met `success`, `response`, `session_id`, `timestamp`.
6. **Errorpad:** exceptions worden gevangen, gemeld via `PrivacyLogger::logSecurityEvent('chatbot_error', ...)` en geven een gebruikersvriendelijke fallback met contactopties.

2.2 GET `/api/chat/welcome` & `/faq`
- Beide endpoints gebruiken hetzelfde responseformat en leveren direct bruikbare quick replies zodat de widget zonder extra logica kan laden.
- `welcome()` deelt de eerste `session_id`; `faq()` fungeert als human-readable menu.

2.3 Hulpmethode `extractIntent`
- Deze methode leest `response['type']` uit zodat logging niet van de detectiefase hoeft af te hangen. Dat voorkomt inconsistentie wanneer responses handmatig worden samengesteld.

3. Frontend widget – `resources/js/chatbot.js`

3.1 Initiatie
- `constructor()` zet state (`isOpen`, `sessionId`, `messageHistory`) en roept `init()`.
- `init()` => `createChatWidget()` (DOM + inline-style injectie), `bindEvents()` (open/close, enter-to-send) en `loadWelcomeMessage()`.
- CSS wordt via `document.createElement('style')` toegevoegd zodat de widget onafhankelijk is van de host-layout.

3.2 Berichtenflow
```js
async sendMessage(text) {
    this.appendMessage({ from: 'user', text });
    this.setTyping(true);

    const payload = { message: text, session_id: this.sessionId };
    const response = await fetch('/api/chat', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload),
    });
```

```
    const data = await response.json();
    this.handleBotResponse(data.response);
}
```

- **Inputbeperkingen:** `maxlength="500"` en realtime teller zorgen dat de front end nooit meer stuurde dan de backend accepteert.
- **Typing indicator:** `setTyping(true)` laat drie lopende puntjes zien tot de backend antwoordt.
- **State update:** `handleBotResponse()` rendert tekst, voegt quick replies toe, toont `action_button` (met `target="_blank"` voor externe links) en slaat alles op in `messageHistory`.

3.3 Foutafhandeling
- Netwerkfouten tonen een rood bannerachtig bericht met “Probeer het opnieuw”; het gebruikersbericht blijft staan zodat herverzenden zinvol is.
- Wanneer de backend `success:false` retourneert, wordt `fallback_response` getoond inclusief quick replies “Contact opnemen” en “Website bezoeken”.

4. Datalaag – belangrijkste modellen en migraties

4.1 `Complaint` model (`app/Models/Complaint.php`)
- **Velden:** titel, beschrijving, `category`, `priority`, `status`, `location`, `lat/lng`, melder-NAW, `internal_notes`, `resolved_at`, `assigned_to`.
- **Kasts:** `lat` en `lng` als `decimal:8` voor nauwkeurige kaartweergave, `resolved_at` als `datetime`.
- **Relaties:** `attachments()`, `notes()`, `statusHistories()`; deze worden in API-resources eager-loaded om N+1 queries te vermijden.
- **Scopes:** `scopeOpen/inBehandeling/opgelost` bieden korte filters voor dashboards; `scopeRecent($days)` gebruikt `now()->subDays($days)` zodat controllers niet zelf datumlogica hoeven te bevatten.

4.2 Migraties die de chatbot direct raken
- `2025_09_22_091228_create_complaints_table.php` legt de basisstructuur vast; `status` is ENUM (`open`, `in_behandeling`, `opgelost`) zodat de chatbot statusteksten met exact dezelfde labels kan tonen.
- `2025_09_22_134511_add_fields_to_complaints_table.php` voegt `category`, `priority`, contactvelden en toewijzing toe – deze data wordt gebruikt in widgets zoals kaarten en notificaties.
- `2025_09_22_091335_create_attachments_table.php`, `091410_create_notes_table.php`, `092030_create_status_histories_table.php` maken het mogelijk om dossierdetails te tonen en een statusgeschiedenis op te bouwen (handig voor chatbot antwoorden over “wat betekent deze status?”).
- `2025_09_23_073641_create_settings_table.php` faciliteert configurabele waarden (bijv. openingstijden) die later ook naar de chatbot kunnen worden doorgeschoven.

5. Routes en rate limiting
- `routes/api.php` groepeert alle publieke endpoints onder `Route::middleware(['throttle:api'])`. Hierdoor delen ze de algemene limiet van 60 req/min.
- Via `AppServiceProvider::configureRateLimiting()` wordt een aparte limiter `chat` toegevoegd:
  ```php
  RateLimiter::for('chat', fn (Request $request) => Limit::perMinute(20)->by($request->ip()));
  ```
  Deze limiter kan op de routes worden gezet (`->middleware('throttle:chat')`) zodat intensieve chatinteracties de rest van de API niet beïnvloeden.

6. Logging, privacy en beveiliging

6.1 Configuratie (`config/logging.php`)

- `privacy_safe` → dagelijkse log (`storage/logs/application.log`, retentie 30 dagen) voor PII-vrije analytics.
- `security` → 90 dagen retentie voor ongebruikelijke gebeurtenissen en fouten.
- `audit` → 365 dagen retentie voor compliance-acties (data export/delete).

6.2 `PrivacyLogger` service
- `logUserAction()` filtert eerst velden waarvan de sleutel mogelijk PII bevat (`email`, `name`, `phone`, `description`, …). Deze worden genegeerd of gehashed zodat ruwe data nooit het log haalt.
- `hashIp()` en `hashUserAgent()` gebruiken `hash('sha256', waarde . config('app.key'))` en nemen de eerste 16 tekens; hierdoor zijn entries niet terug te rekenen zonder de app key.
- `logSecurityEvent()` wordt o.a. in `ChatController` gebruikt bij exceptions, waardoor opsporing van misbruik mogelijk is zonder gevoelige inhoud.
- `logComplaintAction()` plakt het klacht-ID aan logregels zodat auditors kunnen zien welke case is gemuteerd, zonder persoonsgegevens weg te schrijven.

7. Testen en kwaliteitsbewaking
- `tests/Feature/SecurityTest.php` mockt `Log::channel('privacy_safe')` om te garanderen dat privacy logging wordt aangesproken tijdens gevoelige acties (zoals chatbotinteracties).
- `tests/Feature/ComplaintTest.php` controleert of API-responses voor klachten correct zijn opgebouwd, waardoor regressies in de datalaag snel worden gevonden.
- CI scripts in `composer.json`:
  - `composer test` → `php artisan test` (Pest of PHPUnit).
  - `composer code-quality` → draait eerst PHPStan (`level` vanuit `phpstan.neon`) en daarna Laravel Pint voor consistente formatting.

8. Uitbreiden zonder technische schuld
1. **Nieuwe intent toevoegen**
   - Voeg keywords toe aan `$this->intents`.
   - Creëer een `getNieuweIntentInfo()` helper die de responsevorm volgt (`type`, `message`, `quick_replies`, optioneel `action_button`).
   - Breid `knowledgeBase` uit wanneer extra data nodig is.
2. **Data raadpleegbaar maken**
   - Bouw migrations en modelrelaties zodat de chatbot of dashboards dezelfde bron kunnen gebruiken. Houd enums en constants gelijk aan wat de chatbot verstuurt.
3. **Frontend functionaliteit**
   - Voeg UI-features toe in `resources/js/chatbot.js`, compileer met `npm run build` en controleer in meerdere viewports; de widget is volledig self-contained dus wijzigingen raken de rest van de site niet.
4. **Monitoring**
   - Houd `storage/logs/application.log` in de gaten voor trending vragen en gebruik `security.log` om pieken in fouten (bijv. 500 responses) te detecteren. Deze feedback kan worden gebruikt om nieuwe intenten te prioriteren.

Met deze uitgebreide toelichting op de belangrijkste codeonderdelen blijft duidelijk waar functionaliteit leeft en hoe elke laag – service, API, frontend en datamodel – elkaar versterkt in de gemeente-chatbot.