# HopeLine Assistant – Post-Upgrade Fix Pack

Extremely Detailed Implementation Instructions for Replit

READ THIS FIRST
This document is the FINAL FIX PACK for the HopeLine Assistant project.
Replit should follow these instructions CAREFULLY and STEP-BY-STEP to bring the system from “almost done” → “production-ready and sellable”.

Do NOT remove working features. Only extend and refine according to this spec.

Core goals of this Fix Pack:
1) Fully implement pre-intake → appointment integration.
2) Add AI-generated conversation summaries for each appointment.
3) Add robust crisis-handling backend overrides.
4) Upgrade email notifications to include all relevant information.
5) Optionally implement SMS notifications safely.
6) Correct and stabilize analytics (backend + admin UI).
7) Enhance the appointment detail view in admin.
8) Prepare data models for future multi-tenant/super-admin functionality.

Assumptions:
- Project structure: /server, /client, /shared
- Backend: TypeScript/Node (Express or similar)
- Frontend: React, likely with React Query or similar
- Shared models: /shared/schema.ts or equivalent
- Admin routes: /admin/* exists and is functional

---
SECTION 1 – DATA MODEL UPDATES (SHARED SCHEMA)
---

GOAL
Ensure appointments and logs have the fields needed for:
- Pre-intake
- AI summaries
- Per-client support in the future

1.1 Appointment model: required fields
Open the shared schema file (e.g., shared/schema.ts) and ensure the Appointment model contains at least:

- id: string
- clientId: string | null (for now, can default to a single client like “faith_house”)
- name: string
- phone: string | null
- email: string | null
- appointmentType: string (enum-like: "phone_call", "tour", "family_call", etc.)
- contactPreference: string (enum-like: "phone", "text", "email")
- preferredTime: string or Date
- status: string (enum-like: "new", "contacted", "scheduled", "completed", "cancelled")
- notes: string | null
- summary: string | null (AI-generated summary)
- createdAt: Date
- updatedAt: Date

Pre-intake support: choose ONE of the following patterns:

Option A – Flat fields (recommended for now):
- forWho: string | null // "self", "loved_one"
- sobrietyStatus: string | null // "currently_sober", "need_detox", "not_sure"
- financialSupport: string | null // "yes", "not_sure", "no"
- timeline: string | null // "asap", "30_days", "exploring"
- preIntakeNotes: string | null

Option B – JSON field:
- preIntakeData: Json | null // with keys forWho, sobrietyStatus, etc.

Replit should:
- Pick ONE option and apply it consistently in backend and frontend.
- Run any required DB migrations to add missing columns.

1.2 Conversation logs model (for analytics & summaries)
Ensure the conversation log schema includes:

- id: string
- clientId: string | null (optional for now)
- sessionId: string
- role: "user" | "assistant"
- content: string
- category: string | null // "pricing", "availability", "requirements", etc.
- createdAt: Date

If category does not exist, add it. This is used for analytics breakdown.

---
SECTION 2 – PRE-INTAKE FLOW (FRONTEND + BACKEND)
---

GOAL
Collect pre-intake data BEFORE appointment booking and attach it to the appointment so staff can see it in admin.

2.1 Frontend: PreIntakeFlow component

Create a new component if not existing:

- Path: client/src/components/PreIntakeFlow.tsx (or similar)
- Purpose: show a small form that asks:
  - Who is this for? (Self / Loved one)
  - Current sobriety status? (Sober now, Need detox, Not sure)
  - Ability to help with fees? (Yes, No, Not sure)
  - Timeline? (As soon as possible, Within 30 days, Just exploring)
  - Notes (optional long text)

Fields can be simple select inputs or radio buttons.

2.2 Integration with chat / quick actions

In whatever component handles “See if I qualify” or similar quick action in the chatbot:

- When the user clicks that quick action, show the PreIntakeFlow.
- On submit, store this pre-intake data in frontend state and/or

send it to a pre-intake endpoint keyed by sessionId.

Recommended approach:
- Store pre-intake data in React state associated with the current session.
- Also POST it to a backend endpoint like:
  POST /api/preintake
  Body: { sessionId, forWho, sobrietyStatus, financialSupport, timeline, notes }

The backend can:
- Save it in a small pre_intake table keyed by sessionId, OR
- Cache it in memory if applicable (less ideal in serverless environments).

2.3 Backend: attach pre-intake to appointments

When an appointment is created via POST /api/appointments (or equivalent):

- Identify the current sessionId.
- Look up any pre-intake record for that sessionId (or read from request body if sent together).
- Copy the values into the appointment record:
  - If using flat fields: set forWho, sobrietyStatus, etc.
  - If using JSON: set preIntakeData accordingly.
- Save the appointment with the attached pre-intake info.

This ensures that all appointments include context.

2.4 Admin: display pre-intake info

In the admin appointment detail view (in client/src/pages/admin/appointments or similar detail component):

- Add a "Pre-Intake" section with labels and values:
  - For who: Myself / Loved one
  - Sobriety: [value]
  - Financial support: [value]
  - Timeline: [value]
  - Notes: [text or “(none)”]

If no pre-intake data exists, display: “No pre-intake information recorded.”

---
SECTION 3 – AI CONVERSATION SUMMARIES
---

GOAL
Each appointment should have a short, 3–6 sentence AI summary of the conversation so staff can quickly understand the situation.

3.1 Backend: summary generation flow

In the backend logic that handles appointment creation (e.g., server/routes.ts):

After the appointment data is validated and before final save (or immediately after save with an update):

1) Identify sessionId associated with this appointment.
   - This could be passed from the frontend in the appointment creation request or looked up from a session store.

2) Query conversation logs table for that sessionId:
   - Fetch the last ~20 messages (user + assistant), ordered by createdAt ascending.

3) Build a prompt for OpenAI summarization, for example:

"Summarize the following client conversation for a staff member at a sober living home.
Include:
- Who is reaching out (self or loved one, if clear)
- Basic situation and concerns
- How urgent things seem
- What they're hoping for
- Important details for the first phone call.
Keep it 3–6 sentences. Use a neutral, professional tone."

4) Call OpenAI (e.g., gpt-4.1-mini) with this prompt and the messages.

5) Save the result as appointment.summary.

Replit should:
- Use existing OpenAI client utilities if present.
- Ensure API keys are read from env variables.
- Wrap the call in try/catch.

3.2 Failure behavior

If summarization fails (API error, timeouts, etc.):
- Log the error on the server.
- Do NOT block appointment creation.
- appointment.summary may remain null/empty.

3.3 Admin: display the summary

In the admin appointment detail view:

- Add a "Conversation Summary" section:
  - If summary exists: show the text.
  - If none: show "No summary available."

3.4 Email notifications: include summary

Update email notifications (see Section 4) so that when an appointment email is sent to staff, it includes:

- A "Conversation Summary" block with appointment.summary if present.

---
SECTION 4 – EMAIL NOTIFICATIONS (UPGRADE)
---

GOAL
Make sure email notifications are robust and include all useful information: appointment, pre-intake, and summary.

4.1 Identify email sending utility

In server code, find where email is currently sent (Resend, Nodemailer, or other library). If none exist, implement a small email utility module that can send:

- Subject: text
- HTML/text body

- To: staff email address

4.2 Trigger on appointment creation

In the appointment creation backend logic:

- After the appointment has been saved (and ideally after summary, if using synchronous summarization):
  - Call the email send function with a payload that includes:

Subject example:
- “New inquiry for The Faith House from {name}”

Body should include clearly formatted sections:

1) Appointment Info
   - Name: {name}
   - Phone: {phone}
   - Email: {email}
   - Type: {appointmentType}
   - Preferred time: {preferredTime}
   - Status: {status}

2) Pre-Intake Info (if present)
   - For who: {forWho}
   - Sobriety: {sobrietyStatus}
   - Financial support: {financialSupport}
   - Timeline: {timeline}
   - Notes: {preIntakeNotes}

3) Conversation Summary (if present)
   - {summary text}

4) Links
   - A link to the admin appointment detail page if possible (e.g., /admin/appointments?id={appointmentId}).

4.3 Staff email address source

Use one of:
- A global STAFF_EMAIL env variable, OR
- The client’s email stored in a settings table, accessible via /api/settings.

If settings-based, make sure /admin/settings allows editing that email.

4.4 Error handling

If email sending fails:
- Log the error on server side.
- Do NOT fail the HTTP response for appointment creation (user-facing call should still succeed).

---
SECTION 5 – SMS NOTIFICATIONS (OPTIONAL BUT RECOMMENDED)
---

GOAL
Add optional SMS notifications for staff and/or clients without breaking the system if SMS is not configured.

5.1 Twilio (or similar) environment variables

Expect environment variables:

- TWILIO_ACCOUNT_SID
- TWILIO_AUTH_TOKEN
- TWILIO_FROM_NUMBER

5.2 SMS sending utility

Create a small module, e.g., server/sms.ts:

- Exports a function sendSms(to: string, body: string).
- If any Twilio env vars are missing, log a warning and return without throwing.

5.3 Trigger points

On appointment creation:

- If SMS notifications are enabled in settings and staffPhone exists:
  - Send SMS to staff:

  "New inquiry from {name}. Type: {appointmentType}. Preferred: {preferredTime}. Check the admin dashboard for full details."

- If client's contactPreference is "text" and a phone number is provided:
  - Send SMS to client:

  "Hi {name}, this is The Faith House. We've received your request for a {appointmentType}. We'll contact you soon. If anything changes, call or text us at {businessPhone}."

5.4 Failure handling

If Twilio throws an error (invalid phone, rate limits, etc.):
- Catch the error, log it, and do NOT fail the appointment creation process.

---
SECTION 6 – CRISIS HANDLING (BACKEND OVERRIDE)
---

GOAL
Ensure that any message indicating self-harm or intent to harm gets a safe, consistent response, and the AI does NOT continue normally.

6.1 Keyword detection

In the chat route (e.g., POST /api/chat):

Before calling OpenAI, inspect the user message content (lowercased). If it contains any of the following patterns:

- "kill myself"
- "kill myself."
- "want to die"
- "want to die."
- "hurt myself"
- "end my life"
- "suicide"

Then:

- Do NOT call OpenAI.
- Immediately return a canned crisis message, for example:

“I’m really sorry you’re feeling this way. I’m not able to help in an emergency or crisis situation.
Please contact immediate help right now:
- Call or text **988** for the Suicide & Crisis Lifeline
- If you are in immediate danger, call **911** or your local emergency number.
You don’t have to go through this alone.”

6.2 Logging

Log a crisis event in analytics:

- Add a category "crisis_redirect" in conversation logs for that message.

6.3 Frontend display

The chatbot’s message bubble should display this crisis response like any other assistant reply. No further assistant messages should be generated in that turn.

---
SECTION 7 – ANALYTICS CORRECTION & ENHANCEMENT
---

GOAL
Make sure analytics is accurate and useful.

7.1 Logging user AND assistant messages

In the /api/chat route, ensure that both:

- User message (role: "user")
- Assistant response (role: "assistant")

are stored in conversation logs with:

- sessionId
- role
- content
- category (for assistant messages if applicable)
- createdAt

7.2 Category inference

Implement a simple keyword-based classifier for user messages:

- If message mentions "cost", "price", "payment", "fee" → "pricing"
- If mentions "available", "beds", "space" → "availability"
- If mentions "requirements", "rules", "qualify", "qualification" → "requirements"
- If mentions "apply", "application" → "application_process"
- If a crisis keyword matched (see Section 6) → "crisis_redirect"
- Else → "other"

Save category for analytics.

7.3 /api/analytics/summary

Ensure this endpoint returns:

- totalSessions: number of DISTINCT sessionId in conversation logs.
- totalAppointments: count of appointments.

- conversionRate: totalAppointments / totalSessions (0 if sessions == 0).
- messagesByCategory: array of { category: string, count: number }.

Optionally add:
- crisisCount: count where category = "crisis_redirect".

7.4 Admin UI: analytics

In /admin/analytics, display:

- Stat cards for:
  - Total Conversations
  - Total Appointments
  - Conversion Rate
- List or small chart of categories and counts.
- (Optional) show crisisCount prominently.

Handle empty states gracefully (e.g., “No data yet, start chatting to see analytics here.”).

---
SECTION 8 – APPOINTMENT DETAIL VIEW ENHANCEMENT
---

GOAL
Provide a rich, professional appointment detail panel in admin so staff can work efficiently.

8.1 Detail panel layout

In the admin UI for appointments, when clicking a row / “View” button, show:

Sections in this order:

1) Header
   - Name + status badge (New, Contacted, etc.)
   - Appointment type + preferred time

2) Contact Info
   - Phone (click-to-call: tel: link)
   - Email (click-to-email: mailto: link)

3) Pre-Intake
   - For who
   - Sobriety
   - Financial support
   - Timeline
   - Notes

4) Conversation Summary
   - appointment.summary if present.

5) Internal Notes
   - Editable text area for staff-only notes.
   - “Save notes” button triggers PATCH /api/appointments/:id.

6) Actions
   - Quick buttons or a dropdown to change status:
     - New, Contacted, Scheduled, Completed, Cancelled.

8.2 Saving notes and status

Implement a PATCH endpoint:

PATCH /api/appointments/:id

Body can include fields like:
- status
- notes
- preferredTime
- appointmentType

Update the DB record and return the updated appointment for UI refresh.

---
SECTION 9 – CLIENT ADMIN SETTINGS EXPANSION
---

GOAL
Give each client (e.g., Faith House) limited but powerful control over their own settings without exposing super-admin logic.

In /admin/settings (client-facing), allow editing:

- Staff email (for notifications)
- Staff phone (for SMS)
- Appointment types enabled (checkboxes)
- Pre-intake enabled/disabled (toggle)
- After-hours message text
- Business phone number for SMS templates

These settings should be saved in a settings table or document that the backend uses when sending email/SMS and rendering messages.

---
SECTION 10 – TEST PLAN (MUST BE FOLLOWED)
---

GOAL
Verify that all changes work together and nothing is broken.

10.1 Pre-intake + appointment

- Start a chat, go through pre-intake.
- Book an appointment.
- Check admin:
  - Appointment shows.
  - Pre-intake data appears in detail view.

10.2 Summary

- Have a short but realistic conversation.
- Book appointment.
- Check appointment detail:
  - Conversation Summary appears and looks accurate.
- Check staff email:
  - Summary appears in email.

10.3 Analytics

- Use chat for a few different types of questions (pricing, requirements, etc.).
- Create a few appointments.
- Check /admin/analytics:
  - Total conversations > 0.

  - Total appointments > 0.
  - Conversion rate > 0.
  - Categories reflect the test messages.

10.4 Crisis

- In chat, send: “I want to kill myself.”
- Confirm:
  - The response is ONLY the crisis-safe message (988/911).
  - No normal assistant reply.
  - A crisis_redirect entry is logged.

10.5 Email + SMS

- Book a test appointment.
- Confirm email arrives with all sections.
- If SMS configured, confirm staff and/or client SMS arrive.

---
SECTION 11 – DONE CRITERIA
---

The Fix Pack is considered COMPLETE when:

- [ ] Appointment model supports pre-intake + summary (and optional clientId).
- [ ] Pre-intake flow exists, and its values appear in admin appointment details.
- [ ] AI summaries are generated reliably and shown in admin + email.
- [ ] Crisis messages are safely intercepted by backend logic and redirected to 988/911 guidance.
- [ ] Email notifications contain appointment info, pre-intake, and summary.
- [ ] SMS notifications work when configured and never break the system when not configured.
- [ ] Analytics accurately reflect sessions, appointments, conversion, and categories.
- [ ] Appointment detail view is rich, clean, and fully functional with notes and status updates.
- [ ] Client admin settings allow basic control over notifications and intake flow.

Replit should apply this Fix Pack carefully, verifying after each section that tests pass and that existing working functionality remains intact.