Gemini All-in-One Platform — 30-Day PoC Plan & Detailed README

This document expands the existing README into a practical, day-by-day 30-day Proof-of-Concept (PoC) plan with concrete tasks, deliverables, runbook commands, testing, monitoring, stakeholder checklist, risks, and sample code snippets to get the platform from *repo → working PoC → demo-ready MVP*. Use this as a launch playbook for a single full-stack engineer + 1 QA + optional DevOps support. Adjust resources and timing if you have a larger team.

Goal of the 30-Day PoC

- Build a working PoC combining **Google Gemini (web voice + text)** with **JioCX** phone-calling flows.
- Deliver a demo that can:
- Do browser-based voice AI conversations using Gemini.
- Initiate a two-leg phone call using JioCX (your phone → contact) and route audio between parties.
- Show contact management, call queue, call status, and a basic RAG knowledge base for grounded responses.
- Provide docs, tests, and a short recorded demo to share with stakeholders.

Assumptions / Prereqs

- Node.js >= 18 installed
- npm/yarn installed
- Access to Google Gemini API key (Makersuite) or equivalent
- JioCX account + API credentials (for phone-mode testing) — if unavailable, PoC uses mocked JioCX endpoints with a small adapter
- Modern browser (Chrome recommended for WebRTC)
- Optional: ngrok for exposing dev server to remote phone callbacks

---

High-Level Milestones

1. **Days 1–3:** Repo onboarding, environment, and core architecture review
2. **Days 4–9:** Implement Gemini web voice flow (mic → STT → Gemini → TTS → speakers)
3. **Days 10–15:** JioCX integration adapter + real phone call orchestration (or mocked adapter)
4. **Days 16–20:** Contact management, queueing, UI polish, and local persistence
5. **Days 21–25:** RAG knowledge base + grounded search; testing and security review
6. **Days 26–30:** End-to-end testing, demo recording, README update, handoff materials

---

Day-by-Day Plan (Detailed)

Day 1 — Project setup & onboarding

- Tasks:
- Clone repo to `~/projects/gemcall`.
- Install dependencies: `npm install`
- Copy `.env.example` → `.env.local` and add placeholders.
- Run dev server: `npm run dev` and confirm homepage loads.
- Deliverable: Local dev server running with no console errors.

Day 2 — Architecture walkthrough & test harness

- Tasks:
- Map components to features (LiveCall, Chatbot, Queue, Contacts).
- Run test suite: `npm test` — ensure 11/11 tests pass. Fix failing tests.
- Add `DEV_NOTES.md` with architecture diagram and run instructions.
- Deliverable: Passing tests and architecture note created.

Day 3 — Environment and credential gating

- Tasks:
- Implement `config/credentials.ts` that refuses to start phone mode without JIOCX credentials (clear warning)
- Add `.env.local.sample` with example placeholders.
- Deliverable: Safe startup and clear credential warnings.

Day 4 — Gemini API: text request baseline

- Tasks:
- Create `services/geminiService.ts` with a `queryText(prompt)` wrapper.
- Create a small UI on `/playground` to submit text prompts and show responses.
- Add error handling and logs.
- Deliverable: Text-only Gemini request flow working.

Day 5 — STT: capture mic audio and convert to text

- Tasks:
- Add a `useMicrophone` hook that captures audio chunks via Web Audio API.
- Integrate browser-side STT (if Gemini STT is available, call it; else use WebSpeech API for PoC).
- Pipe STT text into `queryText` to get AI response.
- Deliverable: Microphone → STT → Text pipeline operational.

Day 6 — TTS: speak Gemini responses

- Tasks:
- Implement a `playAudioFromText(text)` util using Gemini TTS or Web Speech Synthesis as fallback.
- Wire TTS to response from Gemini so users hear responses.
- Deliverable: Full mic → AI → audio roundtrip in browser.

Day 7 — Web voice flow refinement & UX

• Tasks:
• UI improvements: mic level indicator, start/stop call, connection status badges.
• Unit tests for `useMicrophone` and `geminiService`.
• Deliverable: User-friendly web voice call demo with tests.

Day 8 — Early load and resource limiting

• Tasks:
• Add request throttling and debounce for STT→API calls to avoid rate limits.
• Add metrics logging (console and optional JSON logs).
• Deliverable: Stable usage under demo conditions.

Day 9 — Demo script & quick recording

• Tasks:
• Record a 60–120s demo of web voice flow.
• Prepare short script and sample prompts to demonstrate features.
• Deliverable: Demo video + script.

Day 10 — JioCX: adapter & mock server

• Tasks:
• Create `services/jiocxAdapter.ts` with two modes: `mock` and `real`.
• Implement mock endpoints for call initiation and status callbacks.
• Add `npm run mock-jiocx` to start a small Express server exposing callback endpoints.
• Deliverable: Mock flows ready for UI integration.

Day 11 — Phone UI hookup (initiate call)

• Tasks:
• Add UI to enter "Your phone number" and a contact phone number.
• Call `jiocxAdapter.makeCall(yourPhone, contactPhone)` and show status.
• Deliverable: Initiate call flow using mock adapter.

Day 12 — ngrok + public callbacks

• Tasks:
• Setup ngrok to expose local `mock-jiocx` for callback testing.
• Update callback URL in mock server and app config.
• Deliverable: Verified inbound callback reachability.

Day 13 — Two-leg call orchestration

• Tasks:
• Implement call orchestration flow: call you → connect to contact → bridge audio.
• Simulate bridging in mock by playing sample audio files.
• Deliverable: Simulated two-leg call working in mock.

Day 14 — Real JioCX integration (if credentials available)

• Tasks:

- Swap `jiocxAdapter` to `real` mode and implement real REST calls per JioCX docs.
- Add safe toggles to avoid accidental mass calling.
- Deliverable: Real call initiation works (careful testing only to known numbers).

Day 15 — Call state & logging

- Tasks:
- Add persistent call-logs in local `sqlite` or a JSON file under `data/` for PoC.
- Show call history UI with statuses and durations.
- Deliverable: Call history and logs available.

Day 16 — Contact management & CSV import

- Tasks:
- Implement contacts CRUD and CSV import/export.
- Ensure phone validation and normalizing to E.164 format.
- Deliverable: CSV import & contact CRUD working.

Day 17 — Queue system and reorder

- Tasks:
- Implement drag-and-drop queue using `react-beautiful-dnd` (or built-in simple reorder logic).
- Implement queue start/pause/resume/skip controls.
- Deliverable: Queue management operational.

Day 18 — Access control & environment safety

- Tasks:
- Add a minimal auth gate (dev-mode password) to restrict phone-mode.
- Add `.env.local` check for `ALLOW_PHONE_MODE=true` to avoid accidental calls.
- Deliverable: Safe phone-mode toggles and simple auth.

Day 19 — RAG knowledge base setup

- Tasks:
- Add `services/ragService.ts` to index local docs (small set of markdown or PDF) and run a simple embedding (local library or vector mock).
- Integrate RAG in chatbot responses: if a query matches docs, include grounded snippet.
- Deliverable: Simple RAG-powered responses working.

Day 20 — UI polish & a11y checks

- Tasks:
- Improve UI spacing, status colors, responsive layout.
- Basic accessibility review (labels, keyboard focus, contrast).
- Deliverable: Polished UI for demo.

Day 21 — End-to-end testing

- Tasks:
- Write integration tests around LiveCall component flows (mocking APIs).
- Manual testing checklist and bug triage.

• Deliverable: Test results and bug fix list.

Day 22 — Logging, monitoring & error reporting

• Tasks:
• Add detailed logs for API errors, STT/TTS failures, and JioCX status codes.
• Integrate lightweight error reporting (console + file) for PoC.
• Deliverable: Observability improved.

Day 23 — Security review

• Tasks:
• Ensure `.env.local` never commits; add `npm run check-secrets` to CI-local script.
• Validate phone inputs to prevent header injection and DLT checks for India.
• Deliverable: Basic security hardening done.

Day 24 — Demo polish & acceptance criteria

• Tasks:
• Compile a 5–7 minute demo flow showing both web and phone modes.
• Write acceptance criteria and success metrics (e.g., 95% connection for mock calls, <3s latency for web voice roundtrip).
• Deliverable: Demo checklist and final acceptance criteria.

Day 25 — Documentation & README updates

• Tasks:
• Update README with explicit Quick Start commands, gating, and demo instructions.
• Add `DEMO.md` with script, test phone numbers (mock), and recording checklist.
• Deliverable: Documentation updated.

Day 26 — Demo recording & stakeholder walkthrough

• Tasks:
• Record final demo video (screen + voice) and short README-based walkthrough.
• Create a one-page executive summary describing business value and next steps.
• Deliverable: Demo video and executive summary.

Day 27 — Handoff artifacts & deployment notes

• Tasks:
• Add `DEPLOYMENT.md` covering hosting options (Vercel/Netlify for frontend, Node server for callbacks), environment variables, and ngrok usage.
• Add `ARCHITECTURE.drawio` or PNG for stakeholders.
• Deliverable: Handoff materials prepared.

Day 28 — Final testing and bug fixes

• Tasks:
• Run full test suite and manual regression testing.
• Fix any high-priority issues.
• Deliverable: Stable PoC build.

Day 29 — Soft launch & feedback gathering

- Tasks:
- Share PoC with internal testers (3–5 people) and collect structured feedback.
- Create issue backlog for Phase 2.
- Deliverable: Feedback and backlog.

Day 30 — Retrospective & next-phase plan

- Tasks:
- Run short retrospective, summarize what worked and what didn't.
- Draft a Phase 2 roadmap for scaling to multi-tenant, recording, analytics, and paid plans.
- Deliverable: Retrospective notes and Phase 2 roadmap.

PoC Deliverables (end of Day 30)

- Working web voice demo (mic → Gemini → audio) with recording
- JioCX two-leg calling flow (mocked + optional real integration)
- Contact management and call queue
- RAG-powered chatbot for grounded answers
- Tests, logs, and basic security gating
- Demo video, executive summary, and deployment notes

Sample Commands & Short Runbook

```
# clone and install
git clone <repo-url> gemcall
cd gemcall
npm install

# copy env and fill keys
cp .env.example .env.local
# set GEMINI_API_KEY and (optionally) JIOCX_* keys

# start dev server
npm run dev

# run mock jiocx server (in separate shell)
npm run mock-jiocx

# run tests
npm test

# create demo build
npm run build
npm run preview
```

Testing Checklist

- [ ] Web voice roundtrip latency < 3s
- [ ] STT accurately transcribes simple sentences (>90% demo accuracy)
- [ ] JioCX mock flow completes bridging
- [ ] Call logs persist and show durations
- [ ] Queue pause/resume works reliably
- [ ] RAG returns relevant document snippets

Risks & Mitigations

1. **No JioCX credentials** — use mock adapter + clearly label demo as simulated.
2. **Gemini API limits** — add throttling, caching, and demo-friendly prompt batches.
3. **Audio device permissions** — include clear UX instructions and fallback to text mode.
4. **DLT / regulatory compliance (India)** — for production, require registered Sender ID and DLT entity; PoC avoids mass messaging/calling.

Phase 2 Quick Roadmap (post-PoC)

- Multi-tenant auth, role-based access
- Persistent vector DB for embeddings (Pinecone/Weaviate/Faiss)
- Call recording, transcripts, quality scoring
- Analytics & KPIs dashboard
- Paid integrations (Twilio, Vonage) for international reach
- SLA, monitoring, and infra automation

Appendix: Minimal Example - Gemini text wrapper (TS)

```
// services/geminiService.ts (simplified)
import fetch from "node-fetch";
const GEMINI_KEY = process.env.GEMINI_API_KEY;

export async function queryText(prompt: string){
  const res = await fetch("https://api.gemini.example/v1/generate",{
    method: 'POST',
    headers: { 'Authorization': `Bearer ${GEMINI_KEY}`, 'Content-
Type':'application/json' },
    body: JSON.stringify({ prompt })
  });
  const data = await res.json();
  return data.output || data;
}
```

Note: Replace endpoint with the official Gemini endpoint and follow their SDK docs.

Final Notes

This playbook is intentionally pragmatic: early demos, careful gating for phone functionality, and clear deliverables. If you want, I can:

- Convert this plan into JIRA/Trello cards (exportable CSV)
- Produce the `mock-jiocx` Express server code as a starter
- Build the `useMicrophone` hook and `geminiService` implementation directly into the repo

Tell me which of those you want next and I will add it to the document or create code files.