VocalQ Inbound — Frontend

Complete step-by-step documentation of the dashboard UI, components, and data flow

REACT + TYPESCRIPT + TAILWIND CSS + RECHARTS + VITE

📋 Table of Contents

🖥️ 1. Frontend Overview

The VocalQ frontend is a **React + TypeScript** single-page application that serves as the admin/operator dashboard for the AI voice assistant. It provides:

- **Overview Dashboard** — KPI cards, peak window charts, intent distribution
- **Live Monitor** — Real-time active call tracking with audio visualizers
- **Call Logs** — Searchable, filterable call records with chat-bubble transcript viewer
- **Knowledge Base** — Upload, list, and delete RAG documents

- **Settings** — Manage the AI greeting message

The UI uses a **dark glassmorphism** design language with a mobile-first bottom navigation bar, optimized for both desktop and mobile viewports.

2. Technology Stack

React 18 UI framework	**TypeScript** Type safety	**Tailwind CSS** Utility-first styling	**Vite** Dev server + bundler
Recharts Charts & graphs	**Fetch API** HTTP requests		

3. Folder Structure

```
frontend/
├── App.tsx                    # Main application (routing, state, layout)
├── types.ts                   # TypeScript interfaces and type aliases
├── constants.tsx              # Colors palette + SVG icon components
├── vite.config.ts             # Vite build configuration
├── services/
│   └── api.ts                 # API client (fetch wrapper for backend)
├── components/
│   ├── Sidebar.tsx            # Bottom navigation bar
│   ├── Overview.tsx           # Dashboard overview with charts
│   ├── RealTimeMonitor.tsx    # Live active calls monitor
│   ├── Transcripts.tsx        # Call logs & transcript viewer
│   ├── KnowledgeBase.tsx      # Knowledge base management
│   └── KPICard.tsx            # Reusable KPI metric card
└── docs/
    └── frontend_documentation.html  # This file
```

4. Build & Dev Configuration

Vite Config `vite.config.ts`

The Vite development server and build process are configured as follows:

SETTING	VALUE	PURPOSE
`plugins`	react()	React Fast Refresh + JSX support
`server.host`	`true`	Network-accessible dev server (0.0.0.0)
`server.port`	`5173`	Default Vite port (or configured to 5175)
`resolve.alias`	`@` → `./src`	Path alias for cleaner imports

Start Command

```
cd frontend
npm install
npm run dev
# Opens on http://localhost:5173
```

🎨 5. Design System & Constants

All colors and icons are centralized in `constants.tsx`.

Color Palette

Primary `#8b5cf6`	**Secondary** `#06b6d4`	**Success** `#10b981`	**Warning** `#f59e0b`
Danger `#ef4444`	**Muted** `#64748b`	**Info** `#2563eb`	**Deep** `#4f46e5`

Icon Components

SVG icon components are exported from `Icons` object:

ICON NAME	USED IN	DESCRIPTION
`Icons.Dashboard`	Sidebar → Home	Grid layout icon
`Icons.Analytics`	Overview charts	Chart/bar icon
`Icons.Live`	Sidebar → Live Monitor	Broadcast/signal icon
`Icons.Transcripts`	Sidebar → Call Logs	Document/text icon
`Icons.Settings`	Sidebar → Settings	Gear/cog icon
`Icons.Knowledge`	Sidebar → Knowledge	Book/brain icon

6. TypeScript Type Definitions

All shared types are defined in `types.ts`.

ViewState

```
type ViewState = 'overview' | 'realtime' | 'transcripts' | 'knowledge' | 'settings'
```

Navigation state — determines which main view is currently displayed.

TranscriptPart

```
interface TranscriptPart {
  speaker: 'ai' | 'user';
  text: string;
  timestamp: string;
}
```

Represents one message in a call transcript. Used in chat-bubble rendering.

CallMetric

```
interface CallMetric {
  id: string;
  caller: string;
  timestamp: string;
  duration: number;
  status: string;
  sentiment: string;
  intent: string;
  language: string;
  summary: string;
  transcript: TranscriptPart[];
  token_usage: number;
}
```

Complete call record with all metadata. Used in Call Logs, Live Monitor, and Overview components.

BusinessMetric

```
interface BusinessMetric {
  label: string;
  value: string;
  change: number;
  trend: 'up' | 'down';
}
```

KPI card data structure. Used by the KPICard component and Overview dashboard.

ChartData & AnalyticsData

```
interface ChartData {
  name: string;
  value: number;
}
```

```ts
interface AnalyticsData {
  total_calls: number;
  completed_calls: number;
  missed_calls: number;
  avg_duration: number;
  intent_distribution: Record<string, number>;
  calls_by_hour: ChartData[];
  peak_window: string;
}
```

Analytics payload shape returned from `GET /api/v1/calls/analytics`.

7. API Service Layer

All API calls are centralized in `services/api.ts`.

Base URL

```ts
const API_BASE = 'http://localhost:8000/api/v1'
```

Available Methods

FUNCTION	HTTP REQUEST	RETURNS	USED BY
`api.getAnalytics()`	`GET /calls/analytics`	`AnalyticsData`	Overview.tsx
`api.getCalls()`	`GET /calls/?limit=200`	`CallMetric[]`	Transcripts.tsx
`api.getActiveCalls()`	`GET /calls/active`	`CallMetric[]`	RealTimeMonitor.tsx
`api.getCallDetails(id)`	`GET /calls/{id}`	`CallMetric`	Transcripts.tsx (detail pane)

Note: The Knowledge Base component makes **direct fetch() calls** to `http://localhost:8000/api/v1/admin/knowledge/*` instead of using the centralized API

service.

🏠 8. App.tsx — Main Application

`App.tsx` is the root component that orchestrates the entire UI.

State Management

STATE VARIABLE	TYPE	PURPOSE
`activeView`	`ViewState`	Controls which main view is displayed (overview, realtime, transcripts, knowledge, settings)
`settingsTab`	`string`	Active sub-tab within Settings view (greeting, model, etc.)
`greeting`	`string`	Current AI greeting message (fetched from backend)
`isSaving`	`boolean`	Loading state for greeting save operation
`saveStatus`	`string`	Success/error feedback message

Layout Architecture

```
|   Home   Live   Logs   Knowledge   Settings |
└──────────────────────────────────────────┘
```

View Routing Logic

No React Router is used — views are conditionally rendered based on `activeView`:

```
activeView === 'overview'    → <Overview />
activeView === 'realtime'    → <RealTimeMonitor />
activeView === 'transcripts' → <Transcripts />
activeView === 'knowledge'   → <KnowledgeBase />
activeView === 'settings'    → Inline settings UI (greeting editor)
```

Greeting Feature (Settings View)

1. On mount: fetches current greeting via `GET /admin/settings/greeting`
2. User edits the greeting text in a `<textarea>`
3. On save: sends `POST /admin/settings/greeting` with the new message body
4. Displays success/error feedback with auto-clear timeout

9. Components (Deep Dive)

Navigation (Sidebar) `Sidebar.tsx`

Type: Presentational component

Role: Fixed bottom navigation bar visible on all views. Mobile-first design with a max-width of 480px, centered on screen.

PROPS	TYPE	PURPOSE
`activeView`	`ViewState`	Highlights the active tab
`onViewChange`	`(view: ViewState) ⇒ void`	Callback to switch views

Menu Items

ID	LABEL	ICON	ACCENT COLOR
`overview`	Home	Dashboard	Primary (#8b5cf6)

ID	LABEL	ICON	ACCENT COLOR
`realtime`	Live Monitor	Live	Secondary (#06b6d4)
`transcripts`	Call Logs	Transcripts	Warning (#f59e0b)
`knowledge`	Knowledge	Knowledge	Info (#2563eb)
`settings`	Settings	Settings	Muted (#64748b)

Active tab shows a colored accent line at top and full-opacity icon + label.

Overview `Overview.tsx`

Type: Container component (fetches own data)

Role: Main dashboard — displays KPI cards, peak window bar chart, intent pie chart, and health metrics.

Data Fetching

1. `useEffect` on mount calls `api.getAnalytics()`
2. Transforms response into 4 business stats, intent distribution array, hourly call data, and peak window string

UI Sections

SECTION	LIBRARY	DATA SOURCE
KPI Cards (4x grid)	KPICard component	total_calls, completed, missed, avg_duration
Peak Window Bar Chart	Recharts BarChart	calls_by_hour array, peak_window label
Intent Distribution Pie	Recharts PieChart (donut)	intent_distribution object → array

SECTION	LIBRARY	DATA SOURCE
Health Metrics (4x grid)	Static display	Confidence, Success, Latency, Growth (hardcoded)

KPICard `KPICard.tsx`

Type: Presentational component

Role: Reusable metric card displaying a label, value, and trend indicator.

PROPS (BUSINESSMETRIC)	RENDERED AS
`label`	Small uppercase label text
`value`	Large bold value (e.g., "142", "35s")
`change`	Percentage badge (e.g., "↑ 12%")
`trend`	Green (up) or red (down) badge color

Uses semantic accent colors based on label keywords: "Inbound" → primary, "Answered" → success, "Missed" → danger, "Duration" → secondary.

RealTimeMonitor `RealTimeMonitor.tsx`

Type: Container component with polling

Role: Displays currently active calls in real-time with live duration counters and audio visualizers.

Polling Mechanism

1. On mount: calls `api.getActiveCalls()`
2. Sets up `setInterval` → polls every **3 seconds**
3. Cleanup: clears interval on unmount

Empty State

When no active calls: shows a phone icon, "No Active Calls" message, and a pulsing cyan dot with "Listening for incoming calls..."

Active Call Card Features

- **Caller identity** — phone number and detected language
- **Live duration** — computed from call start timestamp (MM:SS format)
- **Audio Visualizer** — 6-bar animated waveform (CSS keyframe animation)
- **Latest transcript** — shows last transcript line
- **Monitor button** — toggles listen-only mode (green highlight)
- **Control button** — toggles barge-in override mode (red highlight)

AudioVisualizer Sub-Component

Inline component rendering 6 animated bars. Each bar has randomized height and animation delay for a natural waveform effect. Color changes based on monitoring/override state.

Transcripts `Transcripts.tsx`

Type: Container component with master-detail layout

Role: Full call log browser with search, date filters, and a chat-bubble transcript viewer.

State

VARIABLE	TYPE	PURPOSE
`calls`	`CallMetric[]`	All fetched call records
`searchTerm`	`string`	Filter by caller number or intent
`selectedCall`	`CallMetric | null`	Currently selected call for detail pane
`dateFilter`	`string`	all / today / yesterday / week / month / custom
`customDate`	`string`	Selected date when dateFilter is "custom"

Data Loading

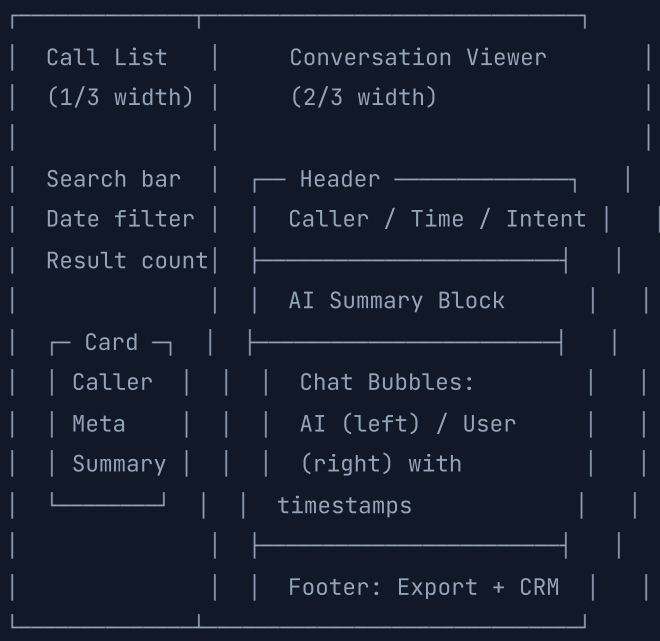
1. On mount: `api.getCalls()` → sorts by timestamp descending → sets `calls`

2 On call select: if transcript is missing, auto-fetches `api.getCallDetails(id)` for full data

Date Filtering

FILTER VALUE	LOGIC
`all`	No date filter
`today`	Matches current date (midnight comparison)
`yesterday`	Matches previous day
`week`	Within last 7 days
`month`	Within last 30 days
`custom`	Shows a date picker, matches selected date

Layout: Master-Detail



ConversationView Sub-Component

- **Header:** Back button, caller number, timestamp (IST), intent badge, sentiment badge, token count
- **Summary Block:** Gradient violet card showing AI-generated call summary

- **Chat Bubbles:** AI messages left-aligned (dark), user messages right-aligned (violet), each with speaker label and timestamp
- **Footer Actions:** "Export Transcript" and "Sync to CRM" buttons (UI only, not yet implemented)

On mobile: the list pane hides when a call is selected; the detail pane becomes full-screen with a back button.

KnowledgeBase `KnowledgeBase.tsx`

Type: Container component (manages own state + API calls)

Role: Upload, view, and delete RAG training documents.

State

VARIABLE	PURPOSE
`docs`	Array of knowledge documents
`loading`	Document list loading state
`uploading`	File upload in-progress state
`error`	Error message (red banner)
`success`	Success message (green banner, auto-clears 5s)

Document Upload Flow

1. User selects a file (.pdf, .txt, .docx) via the drop zone input
2. `handleFileUpload()` creates a FormData and POSTs to `/admin/knowledge/upload`
3. On success: shows green banner ("Successfully uploaded X with Y chunks"), refreshes document list
4. On error: shows red banner with error message

Document List

Fetched from `GET /admin/knowledge/list`. Handles two response formats:

- **Wrapped format:** `{documents: [{doc_id, files: [...]}]}` — flattened to display format
- **Direct array format:** Array of document objects used directly

Each document card shows: file icon (PDF = rose, other = cyan), filename, segment number, preview text, and a delete button.

Document Deletion

Confirm dialog → extracts `doc_id` from composite ID → `DELETE /admin/knowledge/{doc_id}` → refreshes list.

10. Data Flow Diagrams

Overview Dashboard Data Flow

```
App.tsx renders <Overview />
  → Overview useEffect
    → api.getAnalytics()
      → GET http://localhost:8000/api/v1/calls/analytics
        → Backend: calls.py → get_analytics()
          → Queries Supabase "calls" table
          → Computes stats, hourly distribution, peak window
        ← Returns AnalyticsData JSON
    → Transforms data into:
      ├── businessStats[] → KPICard components (4x grid)
      ├── hourWiseCalls[] → Recharts BarChart
      ├── peakWindow      → Text display
      └── intentDistribution[] → Recharts PieChart
```

Live Monitor Data Flow

```
App.tsx renders <RealTimeMonitor />
  → RealTimeMonitor useEffect (polls every 3s)
    → api.getActiveCalls()
      → GET /api/v1/calls/active
        → Backend filters calls where call_status = "active"
```

```
    ← Returns CallMetric[]
  → For each active call:
    ├── Renders call card with caller info
    ├── Computes live duration from timestamp
    ├── Shows AudioVisualizer (animated bars)
    └── Shows latest transcript line
```

Call Logs & Transcript Data Flow

```
App.tsx renders <Transcripts />
  → Transcripts useEffect
    → api.getCalls()
      → GET /api/v1/calls/?limit=200
        → Backend: queries Supabase + joins call_summaries
      ← Returns CallMetric[] (sorted by timestamp)
    → User clicks a call card
      → If transcript missing:
        → api.getCallDetails(call.id)
          → GET /api/v1/calls/{call_id}
          ← Returns full CallMetric with transcript[]
      → Renders ConversationView:
        ├── Summary (gradient violet card)
        └── Chat bubbles (AI left / User right + timestamps)
```

Knowledge Base Data Flow

```
App.tsx renders <KnowledgeBase />
  → KnowledgeBase useEffect
    → fetchDocs()
      → GET /api/v1/admin/knowledge/list
      ← Returns document list (auto-detects format)
    → User uploads file:
      → POST /api/v1/admin/knowledge/upload (FormData)
        → Backend: parse → chunk → embed → store in Qdrant
      ← Returns {success, chunks_created}
      → Refresh document list
    → User deletes document:
```

```
      → DELETE /api/v1/admin/knowledge/{doc_id}
      → Refresh document list
```

Settings (Greeting) Data Flow

```
App.tsx (settings view is inline)
  → useEffect on mount:
    → GET /api/v1/admin/settings/greeting
    ← Returns {greeting: "Welcome to..."}
    → Sets greeting state
  → User edits textarea
  → User clicks "Save Changes":
    → POST /api/v1/admin/settings/greeting
      → Body: {greeting: "updated text"}
    ← Returns success/error
    → Shows feedback toast
```

👁️ 11. User-Facing Views

VIEW	NAV LABEL	COMPONENT	KEY FEATURES
Home	Home	Overview.tsx	4 KPI cards, peak window bar chart, intent pie chart, health metrics grid
Live Monitor	Live Monitor	RealTimeMonitor.tsx	Active call cards with live duration, audio visualizer, monitor/control buttons, 3s polling
Call Logs	Call Logs	Transcripts.tsx	Search + date filter, master-detail layout, chat-bubble transcript, AI summary, export/CRM actions
Knowledge	Knowledge	KnowledgeBase.tsx	File upload (PDF/TXT/DOCX), document list with previews, delete, refresh, success/error banners

VIEW	NAV LABEL	COMPONENT	KEY FEATURES
Settings	Settings	Inline in App.tsx	Greeting message editor with textarea, save button, status feedback