knowledge_demo-customer-portal

Architecture - Customer Contact Portal

> *AI Context: This is a Next.js 14 overlay widget for multi-channel customer contact (voice, chat, video, co-browse).*

Quick Facts

- **Type**: Web Application (Overlay Widget)
- **Stack**: Next.js 14 (App Router), React 18, TypeScript, Tailwind CSS
- **Deployment**: Vercel
- **Key Integrations**:
  - **Airtable**: Configuration & Persona data (per-tenant via Supabase `env_config`; optional)
  - **Surfly**: Co-browsing & Screen sharing
  - **Webex Connect / Contact Center**: Powers Chat, SMS, WhatsApp, Calling, and Callback channels
  - **Webex Instant Connect**: Guest meetings, no need for authentication
  - **Screenshot API**: Background site capture
  - **AWS S3**: Storage for SingleFile captures automatically.
  - **Supabase**: Multi-tenancy, Auth, and Settings storage
- **Last Updated**: 2025-12-04

What This Project Does

A multi-channel contact experience widget that overlays on simulated client websites. Users can initiate contact via call now, video chat, SMS, email, callback request, WhatsApp, or co-browsing. The widget supports persona-based demos with pre-filled contact information.

SaaS / Multi-tenancy: The platform supports multiple tenants. Each user has their own isolated configuration (branding, API keys, webhooks) stored in Supabase. Users can manage their settings via the `/settings` page and view their branded portal at `/user`.

Critical to Know

🚨 Constraints

- **Overlay Architecture**: This is NOT a standard website. No traditional headers/footers. The widget overlays a simulated background site.

- Surfly Window Objects: Co-browse logic requires browser `window` objects. Must be client-side only, cannot be server-rendered.
- Client Components Required: All interactive components must use `"use client"` directive.
- IMI Widget Control: Chat widget controlled by IMI admin portal, not our codebase. Logo/styling changes require IMI portal updates.

🏗️ Architecture Decisions

- View/Container Pattern: Presentational components (view) separated from state management (container) for reusability and testing.
- Component Organization by Feature: Components grouped by functional area (widget, channels, forms, persona, login) not by type.
- Airtable for Config: Persona data and legacy demo config now read per-tenant from Supabase `env_config`; env vars are optional fallbacks. Migration to Supabase DB planned.
- No Storybook Yet: Configuration missing, planned for future.

⚠️ Gotchas

- IMI Widget Hiding: Use specific DOM IDs (`#imi-chatbutton`, `#divchatmain`) with MutationObserver. Generic approaches fail.
- Surfly Client-Side Only: Cannot import Surfly in server components. Use dynamic imports or dedicated client components.
- Persona Routes: Persona login lives at `/user/demo/{locale}/PersonaLogin` (requires auth; uses tenant Airtable keys).
- Screenshot API Token: Optional. Missing token falls back to static background; AWS capture is skipped when AWS creds are absent.

Project Structure

```
next-app/
├── src/
│   ├── app/                    # Next.js App Router
│   │   ├── page.tsx            # Main entry (21 lines, wrapper only)
│   │   ├── layout.tsx          # Root layout
│   │   ├── api/                # API routes
│   │   └── surflyClient.ts     # Surfly integration
│   ├── components/             # React components (by feature)
│   │   ├── channels/          # Channel-specific UI
│   │   ├── forms/             # Form components
│   │   ├── login/             # Login/persona selection
│   │   ├── overlay/          # Background frame
│   │   ├── persona/          # Persona-specific UI
│   │   ├── ui/               # Primitives (icons)
│   │   └── widget/           # Main widget components
│   ├── lib/                  # Utilities & configuration
│   ├── types/                # TypeScript definitions
│   └── contexts/             # React contexts
├── public/                    # Static assets
└── doc-repo/                 # Documentation
```

Key Directories:

- `src/app/`: App Router pages and API routes. Main page is minimal wrapper for Suspense.
- `src/components/`: Organized by feature, not tech. Each folder contains related view and container components.
- `src/lib/`: 7 utility files (constants, webhookUtils, phoneUtils, channelUtils, screenshotUtils, personaUtils, channelConfig).
- `src/contexts/`: PersonaContext for sharing persona data across components.

Key Flows

Authentication / Persona Loading

1. User visits `/user/demo/{locale}`; must be authenticated (Supabase).
2. Tenant config (branding, webhooks, Airtable keys) is pulled from Supabase `tenant_configs.env_config` and fed into runtime config.
3. Persona login flow at `/user/demo/{locale}/PersonaLogin` fetches persona via tenant Airtable keys.
4. Persona data is stored in sessionStorage; PersonaContext provides it to the widget; forms auto-populate.

Contact Channel Interaction

1. User clicks channel in `ContactDrawer` (e.g., "Request Callback")
2. `ContactExperience` shows appropriate form (RequestCallbackFormContainer)
3. Form container manages state, validates input
4. On submit, execute webhook requests to backend
5. Show success/error feedback
6. Close modal, return to channel list

Co-Browse Initiation

1. User clicks "Start Co-Browse" channel
2. Check if Surfly already active (`isSurflyActive()`)
3. If not, call `startVideo()` from surflyClient
4. Surfly SDK creates session, generates PIN
5. Display PIN to user for agent to join
6. Session remains active until user closes browser tab

IMI Chat Widget

1. `IMIWidgetInjector` loads vendor script in client component
2. MutationObserver watches for `#imi-chatbutton` and `#divchatmain` DOM elements
3. When detected, inject CSS classes to hide launcher, position window
4. Custom button triggers chat by accessing iframe internal button: `document.getElementById('imi-chatbutton').contentDocument.getElementById('widgetlbtn').click()`

Integration Points

Airtable

- **Purpose**: Persona data storage and demo configuration (per tenant)
- **Config Source**: Tenant `env_config` in Supabase (falls back to env vars if present); APIs 503 when missing.
- **API Routes**: `/api/login-config` , `/api/persona-by-email` , `/api/personas` , optional logging in `/api/site-capture`
- **Docs**: Airtable API

Supabase

- **Purpose**: Auth, tenant config storage, defaults for env-less deploys.
- **Config**: URL/anon key have baked-in defaults; `tenant_configs.env_config` carries branding/webhooks/Airtable keys.

Surfly (Co-Browse)

- **Purpose**: Screen sharing and co-browsing functionality
- **Config**: Configured in `src/app/surflyClient.ts`
- **Integration**: Client-side only, uses `window.Surfly` object
- **Docs**: Surfly Documentation

IMI / Webex Connect (Chat Widget)

- **Purpose**: Live chat functionality

- **Config**: Widget ID and settings in IMI admin portal (not codebase)
- **Integration**: `IMIWidgetInjector.tsx` loads vendor script
- **Gotcha**: Logo and styling controlled by IMI portal, not our code
- **Docs**: [Webex Connect Documentation]()

AWS S3 (Site Capture)

- **Purpose**: Storage for captured website screenshots
- **Config**: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`, `AWS_S3_BUCKET`
- **Status**: Optional; if missing, archive-status skips S3 and backgrounds fall back to static image.
- **Usage**: Stores HTML/images from the site capture feature when enabled (`NEXT_PUBLIC_ENABLE_SITE_CAPTURE=true`)

Supabase (Multi-tenancy)

- **Purpose**: Authentication, Tenant Configuration, Row-Level Security
- **Config**: `NEXT_PUBLIC_SUPABASE_URL`, `NEXT_PUBLIC_SUPABASE_ANON_KEY`, `SUPABASE_SERVICE_ROLE_KEY`
- **Features**:
  - **Auth**: Email/Password login, Password Reset flow
  - **Database**: `tenant_configs` table with `env_config` JSONB column
  - **RLS**: Ensures users only access their own data
  - **Portal**: `/user` route renders tenant-specific branding

Common Tasks

Adding a New Contact Channel

1. **Add channel config** to `src/lib/channelConfig.tsx`:

   ```
   {
     label: "New Channel",
     icon: <NewIcon />,
     action: { type: 'modal', modalId: 'new-channel' }
   }
   ```

2. **Create content component** in `src/components/channels/NewChannelContent.tsx`:

   ```
   export function NewChannelContent() {
     return <div>Channel UI</div>;
   }
   ```

3. **Update modal router** in `src/components/channels/ChannelDetailModalContainer.tsx`:

   ```
   case 'new-channel':
     return <NewChannelContent />;
   ```

4. **Add icon** to `src/components/ui/icons.tsx` if needed
5. **Test** in both guest and persona modes

Adding a New Persona Form

1. **Create view component** in `src/components/forms/PersonaNewChannelForm.tsx`
2. **Use persona data** from `PersonaContext`:

   ```
   const { persona } = usePersona();
   const prefillData = getRealEmail(persona);
   ```

3. **Add to router** in `PersonaQuickFormDrawer.tsx`
4. **Test** with persona URL parameter

Debugging Surfly Issues

1. **Check browser console** for Surfly SDK errors
2. **Verify script loaded**: Look for `window.Surfly` object
3. **Common issue**: Accessing Surfly in server component → Move to client component
4. **Session stuck**: User must close tab to end session (no programmatic end)

Updating Environment Variables

1. **Local**: Edit `.env.local` (git-ignored)
2. **Vercel**: Update in Vercel dashboard → Settings → Environment Variables
3. **Rebuild**: Redeploy app after changing Vercel env vars
4. **Validate**: Check that variables are accessible in runtime

AI Usage Guide

Best prompts to use:

```
You are an expert Next.js 14 developer working on a multi-channel contact
widget.

Context: This is an OVERLAY WIDGET, not a standard website. It displays over
a simulated background site. All widgets must be client components due to
window object dependencies (Surfly, IMI).
```

```
Current task: <your task>
```

Example queries this doc answers:

- How does persona authentication work?
- Where do I add a new contact channel?
- How is the IMI chat widget integrated?
- What's the view/container pattern for forms?
- Why must Surfly be client-side only?
- Where are environment variables configured?

Component Architecture

Main Components

- **`ContactExperience`**: Orchestrator managing drawer state, channel selection, co-browse
- **`ContactDrawer`**: Sidebar displaying contact channels
- **`ContactLauncher`**: Floating CTA button to open drawer
- **`HomePageContent`**: Handles URL params and background mode
- **`ContactDemoShell`**: Wraps experience with PersonaContext

Form Pattern

View Components (presentation):

- Props: All data and callbacks passed in
- Example: `RequestCallbackForm.tsx`
- Reusable, testable

Container Components (state):

- Manages form state, validation, submission
- Example: `RequestCallbackFormContainer.tsx`
- Wraps view component

Persona Forms (pre-filled):

- Use PersonaContext to auto-populate
- Example: `PersonaCallbackForm.tsx`
- Validate persona data before using

Known Gaps

- [ ] Storybook configuration missing
- [ ] `/demos/login` route referenced but not implemented
- [ ] `NEXT_PUBLIC_SCREENSHOTAPI_TOKEN` not configured
- [ ] Unit test coverage incomplete
- [ ] No PostgreSQL migration plan documented