# **BevGenie Dynamic UI - Context Requirements for Claude**

## **📋 Master Context Document**

This is what you should provide to Claude when asking for help on any task. Copy relevant sections based on what you're working on.

## **1. Project Overview Context**

markdown

# PROJECT: BevGenie AI-Powered Dynamic UI

## What We're Building

BevGenie is a beverage alcohol market intelligence platform. We're adding an AI-powered

conversational interface that dynamically generates personalized UIs based on user questions

and detected personas.

## Current State

- Existing: Static Next.js 14 marketing website with Tailwind CSS

- Tech Stack: React 19, TypeScript, shadcn/ui components

- Hosting: Vercel

- Brand: Professional B2B SaaS for beverage suppliers

## Goal

Transform from static site to conversational AI platform where:

1. Users ask questions in natural language

2. AI detects their persona (sales, marketing, product, executive, researcher)

3. System dynamically generates personalized UI layouts with relevant data

4. Each response is a custom-built interface, not just text

## Key Differentiator

This is NOT a chatbot with text responses. This is conversational UI generation -

the AI creates entire page layouts tailored to the user's role and question.

## **2. Technical Architecture Context**

markdown

# TECHNICAL ARCHITECTURE

## Stack

- **\*\*Frontend\*\***: Next.js 14+ (App Router), React 19, TypeScript

- **\*\*Styling\*\***: Tailwind CSS 4.x, shadcn/ui components

- **\*\*Database\*\***: Supabase (PostgreSQL + pgvector)

- **\*\*Session\*\***: iron-session (cookie-based)

- **\*\*AI\*\***: OpenAI GPT-4o for orchestration, text-embedding-3-small for vectors

- **\*\*Deployment\*\***: Vercel

## Project Structure

```

app/

├── genie/page.tsx # Main AI interface page

├── api/

│ ├── analyze-query/ # Persona detection + context retrieval

│ ├── generate-ui/ # UI specification generation

│ └── session/ # Session management

components/

├── genie/

│ ├── chat-interface.tsx # Conversation UI

│ ├── dynamic-content.tsx # Template renderer

│ └── templates/ # UI templates by persona

lib/

├── supabase/

│ ├── client.ts # DB connection

│ ├── vector-store.ts # Vector operations

│ └── migrations.sql # Schema

├── session/

│ ├── types.ts # Session interfaces

│ ├── config.ts # iron-session config

│ └── session.ts # Session utilities

└── ai/

├── embeddings.ts # OpenAI embeddings

├── persona-detector.ts # Persona classification

└── ui-generator.ts # UI specification generator

```

## Data Flow

1. User enters query → ChatInterface

2. POST /api/analyze-query

- Detect persona using GPT-4o

- Search knowledge base (vector + keyword)

- Update session with persona + message

3. POST /api/generate-ui

- Generate UI spec using GPT-4o

- Save to generated\_pages table

- Return JSON specification

4. DynamicContent receives spec

- Looks up template in componentRegistry

- Lazy loads template component

- Renders with data from spec

## Key Patterns

- Server Components for data fetching

- Client Components for interactivity

- API Routes for AI orchestration

- Template pattern for UI components

- Session-based state management (not localStorage)

## **3. Design System Context**

markdown

# BEVGENIE DESIGN SYSTEM

## Brand Colors

```typescript

const colors = {

deepIndigo: '#0A1930', *// Primary dark, headers*

electricCyan: '#00C8FF', *// Primary accent, CTAs*

refinedCopper: '#AA6C39', *// Secondary accent*

surfaceWhite: '#FFFFFF', *// Backgrounds*

lightDataGray: '#EBEFF2', *// Secondary backgrounds*

charcoalGray: '#333333', *// Body text*

dataGreen: '#198038', *// Success, positive metrics*

riskRed: '#DA1E28', *// Error, negative metrics*

};

```

## Typography

- **\*\*Display Font\*\***: Montserrat (headings, hero, nav)

- **\*\*Body Font\*\***: Inter (paragraphs, UI text)

- **\*\*Sizes\*\***:

- Hero: text-4xl to text-7xl

- Section Headers: text-3xl to text-5xl

- Body: text-base to text-lg

- Small: text-sm to text-xs

## Component Patterns

- Cards: White background, rounded-xl, subtle shadow

- Buttons:

- Primary: bg-[#00C8FF] text-[#0A1930]

- Secondary: bg-[#0A1930] text-[#FFFFFF]

- Inputs: border-[#EBEFF2], rounded-md

- Spacing: Generous padding (p-4 to p-8)

## Layout Principles

- Max width containers: max-w-7xl mx-auto px-4

- Section padding: py-20 md:py-32

- Grid layouts: grid md:grid-cols-3 gap-8

- Responsive: Mobile-first, breakpoints at sm, md, lg

## Voice & Tone

- Professional but approachable

- Data-driven and confident

- Action-oriented ("Get answers" not "Try our tool")

- Industry-specific (use terms like "velocity", "depletions", "SKU")

## **4. Persona System Context**

markdown

# PERSONA DETECTION SYSTEM

## Supported Personas

### 1. Sales Professional

**\*\*Keywords\*\***: competitor, pricing, market share, territory, account, distributor

**\*\*Needs\*\***: Competitive intelligence, account performance, territory gaps

**\*\*UI Preferences\*\***: Dashboards with KPIs, comparison tables, maps

**\*\*Example Questions\*\***:

- "Which territories are underperforming?"

- "Show me competitor pricing in California"

- "Which accounts are growing fastest?"

### 2. Marketing Manager

**\*\*Keywords\*\***: brand, campaign, consumer, trend, social, sentiment

**\*\*Needs\*\***: Campaign ROI, brand health, consumer insights

**\*\*UI Preferences\*\***: Charts, trend lines, brand comparisons

**\*\*Example Questions\*\***:

- "How did our Q4 campaign perform?"

- "What flavors are trending?"

- "Compare brand perception vs competitors"

### 3. Product Manager

**\*\*Keywords\*\***: product, innovation, feature, category, whitespace, SKU

**\*\*Needs\*\***: Product-market fit, innovation opportunities, category trends

**\*\*UI Preferences\*\***: Feature comparisons, innovation timeline, category analysis

**\*\*Example Questions\*\***:

- "What RTD flavors have the most whitespace?"

- "Show me recent product launches in spirits"

- "Which categories are growing?"

### 4. Executive

**\*\*Keywords\*\***: strategy, market, overview, performance, growth, revenue

**\*\*Needs\*\***: High-level insights, strategic direction, performance summary

**\*\*UI Preferences\*\***: Executive dashboards, summary metrics, strategic recommendations

**\*\*Example Questions\*\***:

- "Give me a market overview"

- "What's our overall performance?"

- "Where should we focus next?"

### 5. Researcher

**\*\*Keywords\*\***: data, analysis, research, study, deep dive, metrics, methodology

**\*\*Needs\*\***: Detailed data, methodology transparency, comprehensive analysis

**\*\*UI Preferences\*\***: Data tables, detailed charts, methodology notes

**\*\*Example Questions\*\***:

- "Show me detailed depletions data by state"

- "What's the methodology for market share?"

- "Give me raw data on premium spirits"

## Detection Logic

- **\*\*Primary\*\***: Use GPT-4o with structured output

- **\*\*Fallback\*\***: Rule-based keyword matching

- **\*\*Confidence Threshold\*\***: 0.7 minimum for high confidence

- **\*\*Multi-signal\*\***: Consider query + history + UTM params + behavior

## **5. API Contracts Context**

markdown

# API CONTRACTS

## POST /api/analyze-query

### Request

```typescript

{

query: string; *// User's question (required)*

initMetadata?: boolean; *// Initialize session metadata*

}

```

### Response

```typescript

{

success: boolean;

sessionId: string;

persona: {

type: 'sales' | 'marketing' | 'product' | 'executive' | 'researcher';

confidence: number; *// 0-1*

reasoning: string;

};

context: Array<{ *// From knowledge base*

id: string;

content: string;

similarity: number;

personaTags: string[];

}>;

conversationLength: number;

}

```

## POST /api/generate-ui

### Request

```typescript

{

query: string;

persona: string;

context: Array; *// From analyze-query*

}

```

### Response

```typescript

{

success: boolean;

uiSpecification: {

layout: {

type: 'sales-dashboard' | 'marketing-analytics' | 'comparison-view' | 'data-table';

grid: { columns: number; gap: number };

};

title: string;

components: Array;

data: any;

}>;

theme: {

primaryColor: string;

density: 'compact' | 'comfortable';

};

};

sessionId: string;

}

```

## GET/DELETE /api/session

### GET Response

```typescript

{

success: boolean;

session: {

sessionId: string;

currentPersona: string;

conversationLength: number;

pagesGenerated: number;

duration: number; *// seconds*

};

}

```

## **6. Database Schema Context**

markdown

# DATABASE SCHEMA (Supabase PostgreSQL)

## knowledge\_base

```sql

CREATE TABLE knowledge\_base (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

content TEXT NOT NULL,

embedding vector(1536), *-- OpenAI embeddings*

metadata JSONB DEFAULT '{}',

persona\_tags TEXT[] DEFAULT '{}', *-- ['sales', 'marketing', etc.]*

source\_type VARCHAR(50), *-- 'internal' | 'web' | 'user\_uploaded'*

source\_url TEXT,

created\_at TIMESTAMPTZ DEFAULT NOW(),

updated\_at TIMESTAMPTZ DEFAULT NOW()

);

*-- Indexes for fast search*

CREATE INDEX idx\_embedding ON knowledge\_base USING hnsw (embedding vector\_cosine\_ops);

CREATE INDEX idx\_persona\_tags ON knowledge\_base USING gin (persona\_tags);

```

## sessions

```sql

CREATE TABLE sessions (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

session\_id VARCHAR(255) UNIQUE NOT NULL,

user\_id UUID REFERENCES auth.users(id),

data JSONB NOT NULL, *-- SessionData interface*

created\_at TIMESTAMPTZ DEFAULT NOW(),

updated\_at TIMESTAMPTZ DEFAULT NOW(),

expires\_at TIMESTAMPTZ NOT NULL

);

```

## generated\_pages

```sql

CREATE TABLE generated\_pages (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

session\_id VARCHAR(255) NOT NULL,

persona\_type VARCHAR(50) NOT NULL,

query TEXT NOT NULL,

ui\_specification JSONB NOT NULL,

content\_sources JSONB DEFAULT '[]',

created\_at TIMESTAMPTZ DEFAULT NOW()

);

```

## Vector Search Functions

- `match\_documents(query\_embedding, match\_threshold, match\_count, filter\_personas)`:

Similarity search with persona filtering

- `hybrid\_search(query\_embedding, query\_text, filter\_personas, match\_count)`:

Combined vector + full-text search

## **7. Example Workflows Context**

markdown

# USER WORKFLOWS

## Workflow 1: First-Time User (Sales)

1. User lands on /genie

2. Sees hero: "Ask BevGenie Anything"

3. Clicks suggested prompt: "Which territories are pacing ahead or behind target?"

4. System:

- Detects persona: sales (confidence: 0.85)

- Searches knowledge base for territory/performance data

- Generates sales-dashboard layout

5. User sees:

- Territory map with color-coded performance

- KPI cards: YTD sales, variance, trend

- Table of territories with metrics

- "Top Movers" and "Needs Attention" lists

6. User clicks "New Query" to ask another question

## Workflow 2: Returning User (Marketing)

1. User returns to /genie (session persists)

2. Sees previous conversation in chat

3. Asks: "How is our RTD brand performing vs competitors?"

4. System:

- Remembers persona history (marketing from previous)

- Increases confidence in marketing persona

- Searches for brand/competitor data

- Generates comparison-view layout

5. User sees:

- Side-by-side brand comparison

- Market share charts

- Sentiment scores

- Trend analysis

6. Can continue conversation or start new topic

## Workflow 3: Executive Quick Check

1. User asks: "Give me our Q4 performance summary"

2. System:

- Detects executive persona (high-level question)

- Searches for Q4 performance data

- Generates executive dashboard

3. User sees:

- Summary metrics at top

- Key highlights

- Strategic recommendations

- "Drill down" options for details

## **8. Code Standards Context**

markdown

# CODE STANDARDS

## TypeScript

- **\*\*Strict mode\*\***: Always use strict TypeScript

- **\*\*No any\*\***: Use proper types, create interfaces

- **\*\*Explicit returns\*\***: Always type function returns

- **\*\*Naming\*\***:

- Components: PascalCase

- Functions: camelCase

- Constants: UPPER\_SNAKE\_CASE

- Interfaces: PascalCase with descriptive names

## React/Next.js

- **\*\*'use client'\*\***: Only when needed (interactivity, hooks)

- **\*\*Server Components\*\***: Default for data fetching

- **\*\*Async components\*\***: Use for Server Components

- **\*\*Error boundaries\*\***: Wrap risky operations

- **\*\*Loading states\*\***: Always show feedback

## File Organization

- One component per file

- Colocate related files

- Index files for clean imports

- Separate logic from UI

## Comments

- Why, not what

- Complex logic deserves explanation

- TODO: for future improvements

- FIXME: for known issues

## Testing Approach

- Manual testing checklist

- Test all personas

- Test error states

- Test on multiple devices

## Git Commits

- Prefix with task number: "Task 4.1: Create chat interface"

- Descriptive messages

- Small, focused commits

- Reference issues/tasks

## **9. Environment Setup Context**

markdown

# ENVIRONMENT SETUP

## Required Accounts

1. **\*\*Supabase\*\*** (supabase.com)

- Free tier sufficient for development

- Need: Project URL, Anon Key, Service Role Key

2. **\*\*OpenAI\*\*** (platform.openai.com)

- Pay-as-you-go

- Need: API Key with GPT-4 access

- Models: gpt-4o, text-embedding-3-small

3. **\*\*Vercel\*\*** (vercel.com)

- Connected to GitHub

- Auto-deploy on push

## Local Development

```bash

*# Clone repo*

git clone

cd bevgenie

*# Install dependencies*

npm install

*# Set up environment*

cp .env.example .env.local

*# Fill in all required values*

*# Run database migrations*

*# (Copy SQL from lib/supabase/migrations.sql)*

*# Run in Supabase SQL Editor*

*# Start dev server*

npm run dev

```

## Environment Variables Required

```

NEXT\_PUBLIC\_SUPABASE\_URL=

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=

SUPABASE\_SERVICE\_ROLE\_KEY=

SESSION\_SECRET= # Generate: openssl rand -base64 32

OPENAI\_API\_KEY=

```

## **10. Sample Prompts for Claude**

markdown

# EFFECTIVE PROMPTS FOR CLAUDE

## When Starting a Task

"I'm working on Task X.X: [task name]. Here's the context:

- What we're building: [paste Project Overview]

- Current file structure: [paste relevant structure]

- Dependencies completed: [list task numbers]

- Acceptance criteria: [paste from task list]

Please help me implement this following our [paste Code Standards]."

## For UI Components

"Create the [component name] component following BevGenie design system:

- Colors: [paste Brand Colors]

- Use shadcn/ui components

- Match existing patterns in [reference component]

- Must work with this data structure: [paste interface]"

## For API Routes

"Implement the [endpoint name] API route with:

- Contract: [paste from API Contracts]

- Use these services: [list from lib/]

- Follow this flow: [paste from Data Flow]

- Error handling per [paste standards]"

## For Debugging

"I'm getting [error]. Here's the context:

- Task: [task number]

- Code: [paste relevant code]

- Error: [full error message]

- Expected: [what should happen]

- Architecture: [paste relevant architecture section]"

## For Testing

"Help me create a testing checklist for [component/feature]:

- Personas to test: [paste from Persona System]

- Workflows: [paste relevant workflow]

- Edge cases: [list known issues]

- Acceptance: [paste criteria]"

## **11. Common Pitfalls Context**

markdown

# COMMON PITFALLS TO AVOID

## DON'T

❌ Use localStorage or sessionStorage (not supported in server components)

❌ Make API calls from Server Components (use Server Actions instead)

❌ Import 'use client' components in Server Components unnecessarily

❌ Forget to handle loading states

❌ Skip error boundaries

❌ Use inline styles (use Tailwind classes)

❌ Hardcode API keys in code

❌ Forget to update session on state changes

❌ Mix async/await patterns inconsistently

## DO

✅ Use iron-session for state management

✅ Keep Server Components async for data fetching

✅ Add 'use client' only when needed

✅ Show loading skeletons everywhere

✅ Wrap risky operations in try-catch

✅ Use Tailwind utility classes

✅ Use environment variables

✅ Update session after persona detection

✅ Be consistent with async patterns

✅ Follow TypeScript strict mode

## Known Issues

- Supabase RLS policies can block operations (use service role key for admin)

- OpenAI rate limits (handle 429 errors gracefully)

- Session cookies max size (don't store large data)

- Vector search needs warm-up (first query may be slow)

## **12. Quick Reference Context**

markdown

# QUICK REFERENCE

## File You'll Edit Most

- `components/genie/chat-interface.tsx` - Main user interface

- `lib/ai/persona-detector.ts` - Persona logic

- `lib/ai/ui-generator.ts` - UI generation prompts

- `components/genie/templates/\*` - Add new templates here

- `app/api/\*/route.ts` - API endpoints

## Commands

```bash

npm run dev *# Start dev server*

npm run build *# Test production build*

npm run lint *# Check code quality*

```

## Important URLs

- Local: http://localhost:3000/genie

- Supabase Dashboard: https://app.supabase.com

- Vercel Dashboard: https://vercel.com/dashboard

- OpenAI Usage: https://platform.openai.com/usage

## Quick Debugging

1. Check Supabase logs for DB errors

2. Check Vercel logs for API errors

3. Check browser console for client errors

4. Check Network tab for failed requests

5. Verify environment variables are set

```

---

## \*\*How to Use This Context\*\*

### \*\*For Each Task, Provide:\*\*

1. \*\*Core Context\*\* (always include):

- Section 1: Project Overview

- Section 2: Technical Architecture (relevant parts)

- Section 8: Code Standards

2. \*\*Task-Specific Context\*\*:

- \*\*UI Tasks\*\*: Add Sections 3 (Design System), 5 (Personas)

- \*\*API Tasks\*\*: Add Sections 5 (API Contracts), 6 (Database)

- \*\*AI Tasks\*\*: Add Sections 4 (Personas), 7 (Workflows)

3. \*\*The Specific Task Details\*\*:

```

I'm working on Task [X.X]: [Task Name]

Description: [from task list]

Dependencies: [completed tasks]

Acceptance Criteria: [from task list]

Time Estimate: [from task list]

```

### \*\*Example Complete Prompt:\*\*

```

I'm working on Task 4.1: Create Chat Interface Component

[Paste Section 1: Project Overview]

[Paste Section 2: Technical Architecture - Project Structure]

[Paste Section 3: Design System - Colors, Typography, Component Patterns]

[Paste Section 8: Code Standards]

Task Details:

- Create components/genie/chat-interface.tsx

- Add Textarea for user input with submit button

- Show conversation history with user/assistant messages

- Add suggested prompts section

- Handle loading states

- Call /api/analyze-query and /api/generate-ui

- Trigger onUIGenerated callback

Acceptance Criteria:

- Can send messages and see responses

- Loading states are clear

- Follows BevGenie brand colors

- Mobile responsive

Please implement this component.