Doppel Center v2.0 — Implementation Prompt for Cursor

> **Version:** 1.0 Final
> **Date:** January 2026
> **Author:** C.J. Berno + Claude (Strategic Oversight)

Context & Role

You are acting as **CTO and Lead Engineer** for this project. Your responsibilities:

1. Generate production-ready code changes per phase
2. Flag risks, dependencies, or blockers before they become problems
3. Maintain clean git history with atomic, well-scoped commits
4. Ensure TypeScript types, security best practices, and performance
5. Ask clarifying questions when requirements are ambiguous

Project Overview

Repository: github.com/chrisberno/doppel-center
Production URL: https://doppel.center (Vercel)
Product Name: Doppel Center Enterprise Voice Tech Application

Current State

- v1 code: Vanilla JS + Express (to be archived)
- Local folder structure:

```
doppel-center/
├── original-code/
│   ├── ai-voice-studio-app/   ← New base (Next.js 16)
│   └── doppel-center/         ← v1 code (features to port)
```

Goal

Overwrite v1 with **ai-voice-studio-app** as the new base, then:

1. Port key doppel-center v1 features (multi-provider TTS, IVR exports, voice library)
2. Maintain the existing ai-voice-studio-app capabilities (Chatterbox AI TTS, user auth, projects, payments)
3. Add enterprise Twilio/IVR workflows
4. Refactor UI to Twilio Paste design system
5. Rebrand to "Doppel Center Enterprise Voice Tech Application"

Tech Stack (Inherited from ai-voice-studio-app)

Layer	Technology
Frontend	Next.js 16, TypeScript, Tailwind CSS (→ migrating to Twilio Paste)
Backend	Python 3.11 on Modal (serverless)
Database	Neon (PostgreSQL) + Prisma ORM
Auth	Better Auth
Payments	Polar (keep for credits system)
Storage	AWS S3
New Additions	Twilio SDK, boto3 (direct Polly), @twilio-paste/*

Environment Variables Specification

Create `.env` in `/frontend` with:

```bash
```

```
# Database (Neon)
DATABASE_URL="postgresql://..."

# Auth (Better Auth)
BETTER_AUTH_SECRET="your-secret-min-32-chars"
BETTER_AUTH_URL="http://localhost:3000"

# Payments (Polar) — KEEP from ai-voice-studio-app
POLAR_ACCESS_TOKEN="your-polar-token"
POLAR_WEBHOOK_SECRET="your-polar-webhook-secret"

# AWS (S3 + Polly)
AWS_ACCESS_KEY_ID="your-aws-key"
AWS_SECRET_ACCESS_KEY="your-aws-secret"
AWS_REGION="us-east-1"
AWS_S3_BUCKET_NAME="your-bucket"

# Modal Backend
MODAL_API_URL="your-modal-endpoint"
MODAL_API_KEY="your-modal-key"
MODAL_API_SECRET="your-modal-secret"

# Optional: User-provided at runtime (pass-through, never stored)
# TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN — entered per-request in UI
```

Architecture Decisions (Pre-Implementation)

Decision 1: Twilio Paste + Tailwind

Chosen approach: Hybrid (Option B)

- Use Twilio Paste for **forms, inputs, buttons, modals, alerts**
- Keep Tailwind for **layout utilities** (flex, grid, spacing, responsive)
- Scope Paste components to avoid conflicts
- Full migration is Phase 5; establish patterns in Phase 1.5

Decision 2: TTS Provider Architecture

Chosen approach: Additive (not replacement)

- Keep Chatterbox as default AI provider
- Add Twilio and Amazon Polly as **additional** providers
- User selects provider in UI; credentials passed per-request
- Backend returns uniform `{ audioUrl, duration, provider }` response

Decision 3: Voice Data Source

Chosen approach: Hardcode initially, migrate later

- Port 16+ curated voices from doppel-center v1 to `/lib/voices.ts`
- Add database endpoint in v2.1 for dynamic voice management
- Voice data structure:

```typescript
interface Voice {
  id: string;
  name: string;
  provider: 'chatterbox' | 'twilio' | 'polly';
  language: string;
  languageCode: string;
  gender: 'male' | 'female' | 'neutral';
  sampleUrl?: string;
  description?: string;
}
```

Git Strategy

- **Working branch:** `v2` (orphan branch for clean history)
- **Archive branch:** `archive/v1` (preserve original code)
- **Commit style:** Conventional commits (`feat:`, `fix:`, `chore:`, `refactor:`)
- **Commit frequency:** One commit per logical unit of work (as suggested per phase)

Implementation Phases

Phase 0: Repo Setup & Branding

Prerequisites:

- [ ] Verify Node.js 20.9+ installed (`node --version`)
- [ ] Verify ai-voice-studio-app runs locally (`cd original-code/ai-voice-studio-app/frontend && npm run dev`)
- [ ] Create orphan branch: `git checkout --orphan v2`

Tasks:

1. Copy ai-voice-studio-app to repo root (replacing v1):

```bash
# From repo root
cp -r original-code/ai-voice-studio-app/* .
rm -rf original-code  # Keep only as git history reference
```

2. Update `package.json`:

```json
{
  "name": "doppel-center-enterprise",
  "description": "Enterprise voice tech application with AI TTS, Twilio integration, and IVR exports"
}
```

3. Update `README.md`:
   - Title: "Doppel Center Enterprise Voice Tech Application v2.0"
   - Description: AI voice studio + enterprise Twilio/IVR features
   - Setup instructions for all env vars
   - Architecture diagram
4. Global search/replace:
   - "AI Voice Studio" → "Doppel Center"
   - "ai-voice-studio" → "doppel-center"
   - Update page titles, meta tags, footer
5. Verify app still runs after changes

Commit: `chore: Initial branding for Doppel Center v2`

Phase 1: Backend — Add TTS Providers

File: `backend/text-to-speech/tts.py`

Tasks:

1. Update `requirements.txt`:

```
twilio>=8.0.0
boto3>=1.34.0
```

2. Extend TTS function signature:

```python
@app.function()
@modal.web_endpoint(method="POST")
def generate_speech(
    text: str,
    language: str = "en",
    voice_id: str = None,
    provider: str = "chatterbox",  # NEW: 'chatterbox' | 'twilio' | 'polly'
    # Pass-through credentials (never stored)
    twilio_sid: str = None,
    twilio_auth: str = None,
    aws_access_key: str = None,
    aws_secret_key: str = None,
    aws_region: str = "us-east-1"
) -> dict:
```

3. Implement provider routing:

```python
```

```python
  if provider == "twilio":
    # Use Twilio SDK to synthesize via Polly proxy
    # Requires twilio_sid and twilio_auth
    audio_url = generate_twilio_tts(text, voice_id, twilio_sid, twilio_auth)
  elif provider == "polly":
    # Direct AWS Polly via boto3
    # Requires aws_access_key, aws_secret_key
    audio_url = generate_polly_tts(text, voice_id, aws_access_key, aws_secret_key, aws_region)
  else:
    # Default: Chatterbox (existing implementation)
    audio_url = generate_chatterbox_tts(text, language, voice_id)
```

4. Uniform response format:

```python
  return {
    "success": True,
    "audioUrl": audio_url,
    "provider": provider,
    "voiceId": voice_id,
    "duration": duration_seconds  # if calculable
  }
```

5. Error handling:
   - Validate credentials before API calls
   - Return structured errors: `{ "success": False, "error": "message", "code": "INVALID_CREDENTIALS" }`
   - Fallback to Chatterbox if provider fails and user opts in
6. Deploy to Modal: `modal deploy tts.py`

Commit: `feat(backend): Add Twilio and Amazon Polly providers with credential pass-through`

Phase 1.5: Install Twilio Paste Foundation

Why now: Establishing Paste patterns early prevents rework later.

Tasks:

1. Install packages:

```bash
npm install @twilio-paste/core @twilio-paste/icons @twilio-paste/theme
```

2. Wrap app in Theme Provider (`app/layout.tsx`):

```tsx
import { Theme } from '@twilio-paste/core/theme';

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <Theme.Provider theme="default">
          {children}
        </Theme.Provider>
      </body>
    </html>
  );
}
```

3. Create Paste + Tailwind compatibility layer (`lib/paste-compat.ts`):

```typescript
// Document which utilities come from where
// Paste: forms, buttons, modals, alerts, toasts
// Tailwind: flex, grid, spacing (mx, my, p), responsive (sm:, md:)
```

4. Test that existing UI still renders correctly

Commit: `chore: Install Twilio Paste foundation`

Phase 2: Frontend — Provider Selection & Voice Library

Files:

- `src/app/(protected)/create/page.tsx` (or equivalent TTS form)
- `src/actions/tts.ts`
- `src/lib/voices.ts` (new)
- `src/components/voice-browser.tsx` (new)

Tasks:

1. Create voice data file (`lib/voices.ts`):

```typescript
// Port from original-code/doppel-center/backend/data/voices.json
export const voices: Voice[] = [
  {
    id: "Polly.Joanna",
    name: "Joanna",
    provider: "polly",
    language: "English (US)",
    languageCode: "en-US",
    gender: "female",
    description: "Clear, professional American female voice"
  },
  // ... 16+ voices
];

export const getVoicesByProvider = (provider: string) =>
  voices.filter(v => v.provider === provider);

export const getVoicesByLanguage = (lang: string) =>
  voices.filter(v => v.languageCode.startsWith(lang));
```

2. Add provider selector to TTS form:

```tsx
```

```
// Using Paste Select component
import { Select, Option } from '@twilio-paste/core/select';

<Select id="provider" value={provider} onChange={setProvider}>
  <Option value="chatterbox">AI (Chatterbox)</Option>
  <Option value="twilio">Twilio</Option>
  <Option value="polly">Amazon Polly</Option>
</Select>
```

3. Conditional credential inputs:

```tsx
{provider === 'twilio' && (
  <>
    <Input type="text" placeholder="Account SID" value={twilioSid} onChange={...} />
    <Input type="password" placeholder="Auth Token" value={twilioAuth} onChange={...} />
  </>
)}
{provider === 'polly' && (
  <>
    <Input type="text" placeholder="AWS Access Key" value={awsKey} onChange={...} />
    <Input type="password" placeholder="AWS Secret Key" value={awsSecret} onChange={...} />
    <Select value={awsRegion} onChange={...}>
      <Option value="us-east-1">US East (N. Virginia)</Option>
      <Option value="us-west-2">US West (Oregon)</Option>
      {/* ... */}
    </Select>
  </>
)}
```

4. Create VoiceBrowser component:

```tsx
// Filterable grid/list of voices
// Filters: provider, language, gender
// Each voice card: name, provider badge, sample play button, select button
```

5. Update server action (`actions/tts.ts`):

typescript

```typescript
export async function generateSpeech(formData: FormData) {
  const provider = formData.get('provider') as string;
  const text = formData.get('text') as string;
  const voiceId = formData.get('voiceId') as string;

  // Pass-through credentials (not stored)
  const credentials = provider === 'twilio'
    ? { twilio_sid: formData.get('twilioSid'), twilio_auth: formData.get('twilioAuth') }
    : provider === 'polly'
    ? { aws_access_key: formData.get('awsKey'), aws_secret_key: formData.get('awsSecret') }
    : {};

  const response = await fetch(process.env.MODAL_API_URL, {
    method: 'POST',
    headers: {
      'Modal-Key': process.env.MODAL_API_KEY,
      'Modal-Secret': process.env.MODAL_API_SECRET,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ text, provider, voice_id: voiceId, ...credentials })
  });

  return response.json();
}
```

Commit: `feat(frontend): Provider selector, credential inputs, and voice library`

Phase 3: IVR/Twilio Exports

Files:

- `src/actions/export.ts` (new)
- `src/components/export-modal.tsx` (new)
- Update project detail page

Reference: Port logic from `original-code/doppel-center/backend/services/exportService.js`

Tasks:

1. Create export server action:

```typescript
```

```
export async function generateExport(
  projectId: string,
  format: 'twiml' | 'studio-json' | 'node-snippet' | 'python-snippet'
) {
  const project = await prisma.project.findUnique({ where: { id: projectId } });

  switch (format) {
    case 'twiml':
      return generateTwiML(project);
    case 'studio-json':
      return generateStudioJSON(project);
    case 'node-snippet':
      return generateNodeSnippet(project);
    case 'python-snippet':
      return generatePythonSnippet(project);
  }
}

function generateTwiML(project: Project): string {
  return `<?xml version="1.0" encoding="UTF-8"?>
<Response>
  <Say voice="${project.voiceId}">${escapeXml(project.text)}</Say>
</Response>`;
}

function generateStudioJSON(project: Project): object {
  // Port from doppel-center v1 exportService.js
  return {
    description: "Generated by Doppel Center",
    states: [
      {
        name: "Trigger",
        type: "trigger",
        transitions: [{ next: "say_widget", event: "incomingMessage" }]
      },
      {
        name: "say_widget",
        type: "say-play",
        properties: {
          say: project.text,
          voice: project.voiceId
        }
      }
    ]
  };
}
```

2. Create ExportModal component:

```tsx
import { Modal, ModalHeader, ModalBody } from '@twilio-paste/core/modal';
import { Button } from '@twilio-paste/core/button';

// Dropdown to select format
// Preview pane showing generated code
// Copy to clipboard button
// Download as file button
```

3. Add "Export for Twilio" button to project/audio view

Commit: `feat: Add Twilio IVR exports (TwiML, Studio JSON, snippets)`

Phase 4: Database & Enhancements

Files:

- `prisma/schema.prisma`
- Dashboard page updates

Tasks:

1. Update Prisma schema:

```prisma
```

```
model Project {
  id          String   @id @default(cuid())
  userId      String
  text        String
  audioUrl    String?

  // NEW fields for v2
  provider    String   @default("chatterbox")  // 'chatterbox' | 'twilio' | 'polly'
  voiceId     String?
  voiceConfig Json?    // Store voice settings as JSON
  exportCount Int      @default(0)
  lastExportedAt DateTime?

  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  user        User     @relation(fields: [userId], references: [id])
}
```

2. Run migration:

```bash
npx prisma migrate dev --name add_provider_fields
```

3. Add "Enterprise IVR" section to dashboard:
    - Quick links: Voice Library, Export Templates, Credential Manager (session-only)
    - Recent exports list
    - Provider usage stats
4. Security enhancements:
    - Validate Twilio credentials format (SID starts with "AC", 34 chars; Auth Token 32 chars)
    - Validate AWS credential format
    - Rate limiting on TTS generation
    - Input sanitization for text (prevent XSS in exports)

Commit: `feat: DB enhancements and Enterprise IVR dashboard section`

Phase 5: UI Refactor with Twilio Paste

Approach: Incremental migration, highest-impact pages first

Priority Order:

1. TTS creation form (most user-facing)
2. Voice browser/selector
3. Project list/dashboard
4. Export modal
5. Settings pages
6. Auth pages (if customized)

Tasks per component:

1. Replace form elements:

```tsx
// Before (Tailwind)
<input className="border rounded px-3 py-2" />

// After (Paste)
import { Input } from '@twilio-paste/core/input';
import { Label } from '@twilio-paste/core/label';

<Label htmlFor="text">Script Text</Label>
<Input id="text" type="text" />
```

2. Replace buttons:

```tsx
// Before
<button className="bg-blue-500 text-white px-4 py-2 rounded">Generate</button>

// After
import { Button } from '@twilio-paste/core/button';
<Button variant="primary">Generate</Button>
```

3. Replace layout components:

```tsx
import { Box, Grid, Card } from '@twilio-paste/core';
```

4. Customize theme for enterprise vibe:

```tsx
// Create custom theme extending default
import { CustomizationProvider } from '@twilio-paste/core/customization';

const customTheme = {
  // Twilio brand colors work well for enterprise telecom
};
```

5. Remove/minimize Tailwind conflicts:
    - Keep Tailwind for: `flex`, `grid`, `gap-*`, `p-*`, `m-*`, responsive prefixes
    - Remove Tailwind for: colors, typography, form styling, buttons

Commits:

- `refactor(ui): Migrate TTS form to Twilio Paste`
- `refactor(ui): Migrate voice browser to Twilio Paste`
- `refactor(ui): Migrate dashboard to Twilio Paste`
- `refactor(ui): Migrate export modal to Twilio Paste`

Phase 6: Polish & Vercel Readiness

Tasks:

1. Testing:

```bash
```

```
# Add basic tests
npm install -D vitest @testing-library/react
```

- Test TTS server action with mocked Modal endpoint
- Test export generation functions
- Test credential validation utilities

2. README updates:
    - Complete setup guide with all env vars
    - Architecture diagram
    - API documentation for Modal endpoint
    - Deployment notes for Vercel
3. Vercel configuration:
    - Verify `next.config.js` has no issues
    - Ensure all env vars are in Vercel dashboard
    - Test preview deployment
4. Performance:
    - Add loading states for TTS generation
    - Optimize voice library (virtualized list if needed)
    - Image optimization for voice avatars (if any)
5. Final cleanup:
    - Remove any remaining ai-voice-studio-app branding
    - Remove unused dependencies
    - Fix TypeScript strict mode errors (if any)

Commit: `chore: Final polish, tests, and Vercel optimizations`

Success Criteria

- [ ] App deploys successfully to Vercel
- [ ] User can generate TTS with all three providers

- [ ] Credentials are never stored (pass-through only)
- [ ] Voice library shows 16+ voices with filters
- [ ] Export modal generates valid TwiML, Studio JSON, code snippets
- [ ] UI uses Twilio Paste for core components
- [ ] All branding shows "Doppel Center"
- [ ] README is comprehensive and accurate

Reference Files

When implementing, reference these from `/original-code/doppel-center/`:

Feature	Source File
Voice data	`backend/data/voices.json`
Export logic	`backend/services/exportService.js`
TTS providers	`backend/services/providers/twilio.js`
Voice registry	`backend/services/voiceRegistry.js`
Frontend voice UI	`frontend/js/app.js`

Questions for Product Owner

Before starting implementation, clarify:

1. **Voice samples:** Should we include audio sample URLs for each voice? If yes, where are they hosted?
2. **Credential storage preference:** Any interest in optional encrypted credential storage for convenience (v2.1 feature)?
3. **Polar payments:** Keep credits system as-is, or modify pricing for enterprise providers?
4. **Custom domain:** Is doppel.center staying, or moving to a subdomain structure?

Ready to begin Phase 0. Awaiting confirmation to proceed.