



**softlTo 4. DÖNEM 1. GRUP**  
**FRONTEND DEVELOPER PROGRAMI**

**CEX: AI ÇAĞRI ASİSTANI**  
**BİREYSEL BİTİRME PROJESİ RAPORU**

Senaryo ve Kural Tabanlı Otomatik Çağrı Platformu

**Okan ŞAHİN**

Temmuz, 2026  
softlTo Yazılım Bilişim Akademisi, İstanbul

# İÇİNDEKİLER

|                                                            |    |
|------------------------------------------------------------|----|
| 1. GİRİŞ .....                                             | 3  |
| 1.1. Projenin Amacı .....                                  | 3  |
| 1.2. Proje Kısıtları .....                                 | 3  |
| 2. MATERYAL VE YÖNTEM .....                                | 4  |
| 2.1. Frontend: React, Vite, TypeScript, Tailwind CSS ..... | 4  |
| 2.2. Backend: Node.js, Express, WebSocket .....            | 4  |
| 2.3. Veritabanı: PostgreSQL .....                          | 5  |
| 2.4. Yapay Zeka: Hugging Face Inference API .....          | 5  |
| 2.5. Metinden Sese: ElevenLabs .....                       | 6  |
| 2.6. Kimlik Doğrulama: JWT .....                           | 6  |
| 3. SİSTEM MİMARİSİ .....                                   | 7  |
| 4. KURULUM VE YAPILANDIRMA .....                           | 8  |
| 4.1. Bağımlılıkların Kurulumu .....                        | 8  |
| 4.2. Ortam Değişkenleri .....                              | 8  |
| 4.3. Veritabanı Şeması .....                               | 9  |
| 4.4. Geliştirme Sunucusunun Başlatılması .....             | 9  |
| 4.5. Kayıt Olma ve Giriş Yapma .....                       | 10 |
| 4.6. Senaryo Oluşturma .....                               | 10 |
| 4.7. Test Araması Başlatma .....                           | 11 |
| 5. MALİYET VE PERFORMANS OPTİMİZASYONU .....               | 12 |
| 6. DEPLOY (DAĞITIM) .....                                  | 13 |
| 7. SONUÇ .....                                             | 14 |
| 8. KAYNAKÇA .....                                          | 15 |

# 1. GİRİŞ

Bu bölümde projenin amacı ve projede karşılaşılan/benimsenen kısıtlara yer verilmiştir.

## 1.1. Projenin Amacı

Bu projenin amacı, önceden tanımlanmış senaryolara ve kural setlerine göre otomatik olarak giden arama yapabilen veya gelen aramaları cevaplayabilen, tamamen Türkçe arayüzlü bir yapay zeka çağrı platformu (CEX) geliştirmektir. Platform, bir işletmenin (örneğin randevu hatırlatma yapan bir klinik) çağrı akışlarını insan müdahalesi olmadan yönetebilmesini hedefler: sistem, kullanıcının tanımladığı bir *senaryo* (sistem promptu, karşılama mesajı, kural seti ve toplanacak değişkenler) doğrultusunda konuşmayı yürütür, konuşmayı metne çevirir (STT), gerekli durumlarda büyük dil modeline (LLM) danışarak doğal bir yanıt üretir, bu yanıtı sese dönüştürür (TTS) ve tüm çağrı geçmişini kalıcı olarak kaydeder.

Çalışma; konuşma tanıma, doğal dil üretimi ve metinden sese sentezleme gibi yapay zeka bileşenlerinin gerçek zamanlıya yakın (tur bazlı) bir web uygulaması içinde nasıl bir araya getirilebileceğini, bunun yanında çok-adımlı bir kural motorunun büyük dil modeli çağrılarının maliyetini nasıl azaltılabileceğini göstermeyi amaçlamaktadır.

## 1.2. Proje Kısıtları

**Kısıt 1:** Ses akışı gerçek zamanlı (sürekli, iki yönlü) değil, tur bazlıdır — kullanıcı bir düğmeyi basılı tutarak konuşur, bırakınca kayıt işlenir ve yapay zeka yanıt verir ("bas-konuş-bırak" modeli). Kesintisiz, eş zamanlı ses akışı (barge-in) kapsam dışıdır.

**Kısıt 2:** Kimlik doğrulama basit bir erişim kapısı olarak tasarlanmıştır — e-posta/şifre ile kayıt ve JWT tabanlı giriş bulunur, ancak kullanıcı başına veri izolasyonu yoktur; giriş yapan tüm kullanıcılar aynı paylaşılan senaryo/arama verisini görür. Şifre sıfırlama ve e-posta doğrulama kapsam dışıdır.

**Kısıt 3:** Gerçek telefon şebekesi (PSTN) üzerinden arama yapılması (Twilio entegrasyonu) mimari olarak hazırlanmış ancak uçtan uca test edilmemiştir; bu, kullanıcının kendi Twilio hesabı, satın alınmış bir telefon numarası ve herkese açık bir webhook adresi gerektirir. MVP kapsamında varsayılan ve tam test edilen mod, tarayıcı üzerinden mikrofona yapılan simülasyon modudur.

**Kısıt 4:** Rapor, projenin mimarisi, kurulum adımları ve alınan tasarım kararlarıyla sınırlıdır; İK, pazarlama veya işletme modeline dair konular kapsam dışı bırakılmıştır.

## 2. MATERYAL VE YÖNTEM

### 2.1. Frontend: React, Vite, TypeScript, Tailwind CSS

**React**, kullanıcı arayüzü oluşturmak için kullanılan, bileşen (component) tabanlı bir JavaScript kütüphanesidir. Arayüzün her parçası (Panel, Senaryolar, Ayarlar vb.) bağımsız, yeniden kullanılabilir bileşenler olarak yazılır. **Vite**, geliştirme sırasında anında yenileme (Hot Module Replacement) sağlayan ve üretim için optimize edilmiş statik dosyalar üreten bir derleme aracıdır. **TypeScript**, JavaScript'e statik tip denetimi ekleyerek, özellikle API sözleşmelerinin (örneğin bir senaryonun alanları) frontend ve backend arasında tutarlı kalmasını derleme zamanında garanti eder. **Tailwind CSS**, önceden tanımlı yardımcı sınıflarla (utility classes) hızlı ve tutarlı arayüz stillemesi sağlayan bir CSS çatısıdır.

Sayfa yönlendirmesi `react-router-dom` ile, sunucu verisinin önbelleklenmesi ve senkronizasyonu ise `@tanstack/react-query` ile yönetilmektedir.

### 2.2. Backend: Node.js, Express, WebSocket

**Node.js** üzerinde çalışan **Express** çatısı, REST API uç noktalarını (senaryolar, aramalar, ayarlar, kimlik doğrulama) sunar. Anlık, çift yönlü iletişim gerektiren çağrı döngüsü için ham **WebSocket** protokolü (`ws` paketi) kullanılır: tarayıcı mikrofon kaydını taban64 (base64) olarak sunucuya gönderir, sunucu konuşmayı metne çevirip yapay zeka yanıtını sentezleyip aynı bağlantı üzerinden geri yollar.

Üretimde tek bir Express süreci hem API/WebSocket'i hem de derlenmiş React uygulamasını aynı porttan sunar; bu, ayrı bir statik barındırma servisi veya CORS yapılandırması gerektirmeden tek bir servis olarak dağıtımı mümkün kılar.

### 2.3. Veritabanı: PostgreSQL

Tüm kalıcı veri (kullanıcılar, senaryolar, aramalar, çağrı turları, ayarlar) **PostgreSQL** içinde saklanır. `node-postgres` (`pg`) kütüphanesi ile parametrelili SQL sorguları kullanılır; senaryoların kuralları gibi iç içe geçmiş yapılar `JSONB` sütunlarında tutularak hem esneklik hem de sorgulanabilirlik sağlanır. Uygulama açılışında şema, `CREATE TABLE IF NOT EXISTS` deyimleriyle kendiliğinden oluşturulur; bu sayede ayrı bir migrasyon aracına ihtiyaç duyulmaz.

### 2.4. Yapay Zeka: Hugging Face Inference API

Konuşmadan metne çevirme (STT) ve doğal dil üretimi (LLM), resmi `@huggingface/inference` istemcisi aracılığıyla **Hugging Face Inference Providers** üzerinden yapılır. STT için `openai/whisper-large-v3`, LLM için `Qwen/Qwen2.5-7B-Instruct` modelleri kullanılmaktadır. LLM'e her turda yalnızca son birkaç turluk kaydırmalı bir geçmiş penceresi gönderilir ve modelden katı bir JSON şablonuyla (yanıt metni, konuşmanın bitip bitmediği, toplanan

değişkenler) yanıt vermesi istenir; bu sayede kural motoru ile yapay zeka çıktısı aynı sözleşme üzerinden birleştirilir.

Geliştirme sürecinde, HF Inference Providers kataloğunda canlı (aktif) durumda olan metinden sese (TTS) modelleri araştırılmış; güvenilir Türkçe desteği sunan bir model bulunamadığı tespit edilmiştir (bkz. 2.5).

## 2.5. Metinden Sese: ElevenLabs

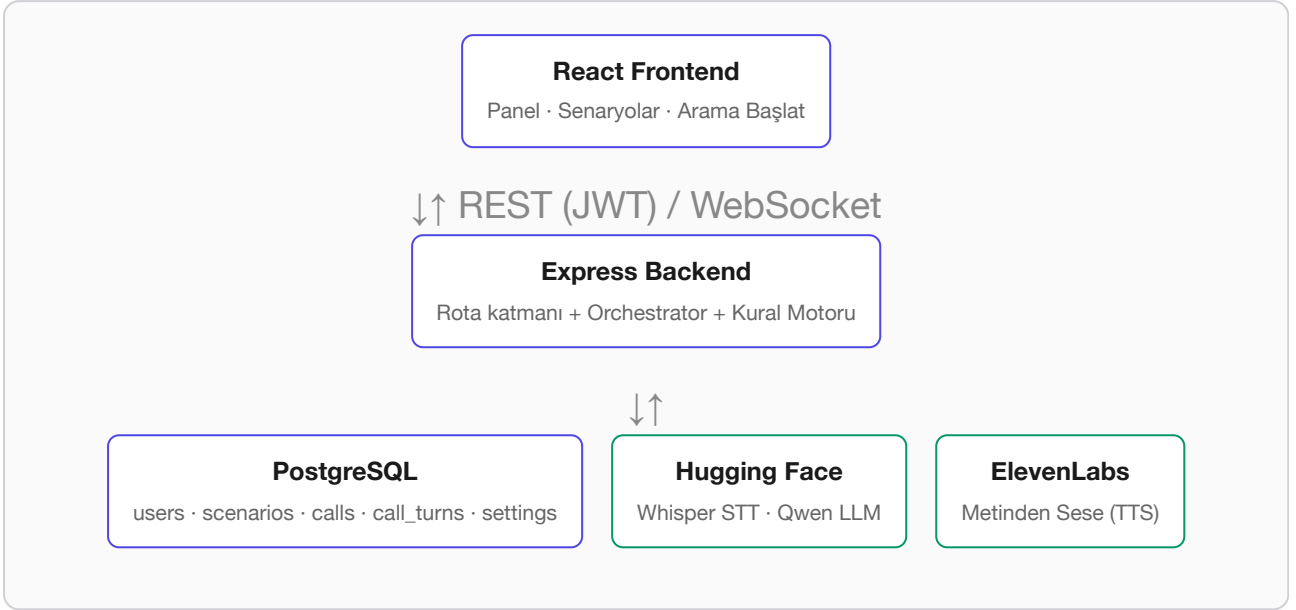
Proje sırasında yapılan araştırma, Hugging Face üzerinde canlı durumdaki metinden sese modellerinin (örn. `ResembleAI/chatterbox`) yalnızca İngilizce'ye sabit varsayılan bir uca yönlendirildiğini; modelin gerçek çok dilli/Türkçe modunun ise Hugging Face'in kayıtlı olmadığı ayrı bir sağlayıcı rotasında bulunduğunu ortaya koymuştur. Bu nedenle metinden sese sentezi için, Türkçe dahil 29 dili resmi olarak destekleyen **ElevenLabs** (`eleven_multilingual_v2` modeli) doğrudan entegre edilmiştir; STT ve LLM adımları Hugging Face üzerinde kalmıştır.

## 2.6. Kimlik Doğrulama: JWT

Kullanıcı kayıt ve girişi, şifrelerin `bcryptjs` ile tuzlanarak (salt) özetlenmesi ve oturumların durum bilgisi tutulmayan (stateless) **JSON Web Token (JWT)** ile temsil edilmesi üzerine kuruludur (`jsonwebtoken` paketi). Jeton, tarayıcıda `localStorage`'da tutulur; her REST isteğine `Authorization: Bearer` başlığı olarak eklenir. Tarayıcı WebSocket bağlantılarına özel başlık ekleyemediği için, WebSocket bağlantılarında aynı jeton bir sorgu parametresi (`?token=`) olarak taşınır ve bağlantı yükseltmeden (upgrade) önce sunucu tarafında doğrulanır.

### 3. SİSTEM MİMARİSİ

Aşağıdaki şema, bir çağrı sırasında bileşenler arasındaki veri akışını özetlemektedir.



Şekil 3.1. CEX bileşenler arası veri akışı

Bir çağrı turunda sırasıyla: (1) tarayıcı mikrofon kaydını WebSocket üzerinden sunucuya gönderir, (2) sunucu Hugging Face Whisper ile konuşmayı metne çevirir, (3) kural motoru metni tanımlı anahtar kelimelerle eşleştirmeye çalışır — eşleşme varsa LLM hiç çağrılmadan önceden tanımlı bir yanıt kullanılır (maliyet optimizasyonu), eşleşme yoksa Qwen LLM'den bir yanıt istenir, (4) üretilen yanıt metni ElevenLabs ile sese dönüştürülür, (5) hem transkript hem de kullanım sayaçları (STT/LLM/TTS çağrı adedi, tahmini token) PostgreSQL'e yazılır ve ses/metin tarayıcıya geri gönderilir.

## 4. KURULUM VE YAPILANDIRMA

### 4.1. Bağımlılıkların Kurulumu

Proje bir monorepo yapısındadır: React uygulaması depo kökünde, Express sunucusu ise kendi `package.json`'ına sahip `server/` alt dizinindedir. Kök dizinden çalıştırılan tek bir komut ikisini de kurar:

```
npm run install:all
```

### 4.2. Ortam Değişkenleri

`server/.env.example` dosyası `server/.env` olarak kopyalanır ve aşağıdaki değerler doldurulur:

| Değişken                        | Amaç                                                 |
|---------------------------------|------------------------------------------------------|
| <code>HF_TOKEN</code>           | Hugging Face erişim jetonu (STT + LLM)               |
| <code>ELEVENLABS_API_KEY</code> | ElevenLabs API anahtarı (TTS)                        |
| <code>DATABASE_URL</code>       | PostgreSQL bağlantı adresi                           |
| <code>JWT_SECRET</code>         | Oturum jetonlarını imzalamak için rastgele bir değer |

### 4.3. Veritabanı Şeması

Uygulama ilk kez ayağa kalktığında aşağıdaki tablo tanımları otomatik olarak uygulanır:

```
CREATE TABLE IF NOT EXISTS users (  
  id UUID PRIMARY KEY, email TEXT UNIQUE, password_hash TEXT, created_at TIMESTAMPTZ  
);  
CREATE TABLE IF NOT EXISTS scenarios ( ... kurallar JSONB, toplanacak_degiskenler  
JSONB ... );  
CREATE TABLE IF NOT EXISTS calls ( ... sonuc JSONB, kullanim JSONB ... );  
CREATE TABLE IF NOT EXISTS call_turns ( ... );  
CREATE TABLE IF NOT EXISTS settings ( id INTEGER PRIMARY KEY DEFAULT 1 CHECK (id =  
1), ... );
```

### 4.4. Geliştirme Sunucusunun Başlatılması

```
npm run dev
```

Bu komut frontend'i (`localhost:5173`) ve backend'i (`localhost:4000`) birlikte başlatır; Vite geliştirme sunucusu `/api` ve `/ws` isteklerini otomatik olarak backend'e yönlendirir.

### 4.5. Kayıt Olma ve Giriş Yapma

Uygulamanın tamamı girişe kapalıdır. İlk kullanımda **Kayıt Ol** sayfasından bir hesap oluşturulur; hızlı test için "Demo Kullanıcı" düğmesi sabit bir demo hesabıyla giriş yapmayı, hesap yoksa otomatik olarak oluşturup girişe devam etmeyi sağlar.

Şekil 4.5.1. Giriş Yap ekranı (stilize temsil)

## 4.6. Senaryo Oluşturma

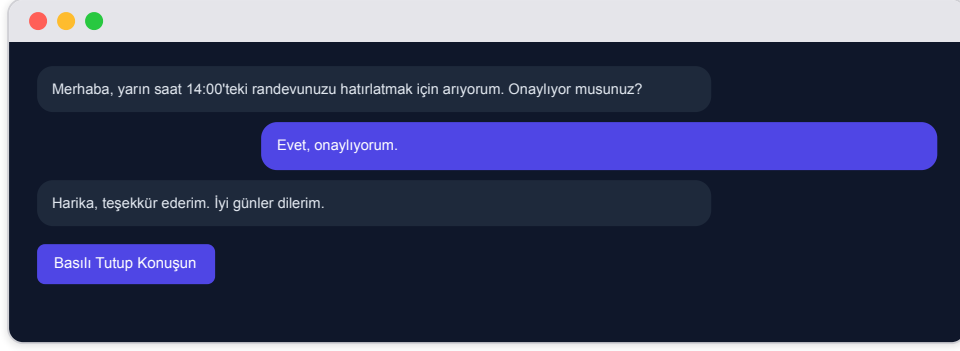
Senaryolar sayfasından yeni bir senaryo oluşturulurken; senaryo adı, yön (giden/gelen/her ikisi), sistem promptu, karşılama mesajı, toplanacak değişkenler ve isteğe bağlı kurallar (anahtar kelime → sonlandır / özel yanıt / değişken kaydet) tanımlanır.

Şekil 4.6.1. Senaryo oluşturma ekranı (stilize temsil)

## 4.7. Test Araması Başlatma

Arama Başlat sayfasından bir senaryo seçilip arama başlatıldığında, yapay zekanın karşılama mesajı otomatik olarak seslendirilir; kullanıcı mikrofona düğmesini basılı tutup konuştuğunda konuşma metne çevrilir, kural motoru veya LLM bir yanıt üretir ve bu yanıt sesli olarak çalınır.





Şekil 4.7.1. Aktif çağrı ekranı — turn-based konuşma akışı (stilize temsil)

Not: Yukarıdaki arayüz temsilleri, projenin gerçek bileşen yapısına ve renk paletine sadık kalınarak oluşturulmuş stilize gösterimlerdir; proje kapsamında tarayıcı üzerinden ekran görüntüsü alınmamıştır (bkz. Kısıt notları).

## 5. MALİYET VE PERFORMANS OPTİMİZASYONU

Her yapay zeka API çağrısı bir maliyet/kota tükettiğinden, orchestrator katmanı (`server/src/services/orchestrator.ts`) aşağıdaki önlemleri uygular:

- **Kural kısa devresi:** Kullanıcının konuşmasında tanımlı bir anahtar kelime eşleşirse ve eylem "özel yanıt" veya "sonlandır" ise, LLM hiç çağrılmadan önceden yazılmış yanıt kullanılır.
- **Kaydırmalı geçmiş penceresi:** LLM'e her turda yalnızca son ~6 turluk konuşma geçmişi gönderilir; bu, çağrı uzadıkça istek boyutunun (ve maliyetinin) sınırsız büyümesini engeller.
- **Sabit token sınırı:** LLM çağrılarında `max_tokens` düşük bir değerde sabitlenmiştir; yanıtlar kısa, konuşma diline uygun tutulur.
- **TTS ses önbellesi:** Aynı metin (örn. karşılama mesajı) tekrar seslendirileceğinde, ElevenLabs'e yeniden istek atmak yerine önbellekteki ses kullanılır.
- **Sessizlik filtresi:** Tarayıcı tarafında, çok kısa/sessiz ses kayıtları hiç sunucuya gönderilmeden elenir.
- **Tur limiti:** Ayarlar sayfasından yapılandırılabilen bir üst sınıra ulaşıldığında, çağrı LLM çağrılmadan otomatik ve nazikçe sonlandırılır.
- **Atomik kullanım kaydı:** Bir turun ortasında (örn. TTS aşamasında) bir hata oluşursa, o ana kadar gerçekleşmiş olan STT/LLM çağrıları kullanım sayaçlarına yine de işlenir ve arama "başarısız" olarak işaretlenir — kısmi bir hata, kullanım verisinin kaybolmasına veya aramanın sonsuza dek "devam ediyor" durumunda asılı kalmasına yol açmaz.

## 6. DEPLOY (DAĞITIM)

Uygulama tek bir servis olarak dağıtılır: derlenmiş Express sunucusu hem API/WebSocket'i hem de derlenmiş React uygulamasını ( `dist/` ) aynı porttan sunar.

```
npm run build:all    # frontend + backend derlenir
npm start            # server/dist/index.js, tek porttan her şeyi sunar
```

Kullanılan barındırma: **Render** (uygulama servisi) + **Supabase** (PostgreSQL). Deploy sürecinde karşılaşılan somut bir sorun ve çözümü:

**Sorun:** İlk deploy denemesinde sunucu `ENETUNREACH` hatasıyla çöktü. Kök neden araştırıldığında, Supabase'in "Direct connection" adresinin yalnızca IPv6 (AAAA) kaydına çözümlendiği, Render'ın ise giden IPv6 desteği olmadığı tespit edildi.

**Çözüm:** Supabase panelinden "Direct connection" yerine yalnızca IPv4 üzerinden çalışan **Session Pooler** bağlantı adresi kullanılarak `DATABASE_URL` güncellendi; sorun ek bir kod değişikliği gerekmeden çözüldü.

Veritabanı artık PostgreSQL olduğundan, barındırma platformunun dosya sistemi geçici (ephemeral) olsa dahi veriler redeploy/restart'larda kaybolmaz.

## 7. SONUÇ

Bu çalışmada, senaryo ve kural tabanlı bir yapay zeka çağrı asistanı olan CEX'in uçtan uca tasarımı ve geliştirilmesi gerçekleştirilmiştir. React/TypeScript tabanlı bir frontend, Express/PostgreSQL tabanlı bir backend, Hugging Face üzerinden konuşma tanıma ve dil üretimi, ElevenLabs üzerinden Türkçe metinden sese sentezi ve JWT tabanlı basit bir kimlik doğrulama katmanı bir araya getirilmiştir.

Geliştirme sürecinde, "modeli Hugging Face üzerinde bulundur" şeklindeki başlangıç kısıtı ile "arayüz tamamen Türkçe olsun" hedefinin çatıştığı somut bir nokta ortaya çıkmıştır: canlı durumdaki Hugging Face TTS modelleri arasında güvenilir Türkçe desteği sunan bir seçenek bulunamamış, bu nedenle yalnızca metinden sese bileşeni için ElevenLabs'e geçilmiş, konuşma tanıma ve dil modeli Hugging Face üzerinde tutulmuştur. Bu, bir teknoloji seçiminin önce varsayılmaması, gerçek API kataloğu üzerinden doğrulanması gerektiğini gösteren somut bir örnektir.

Ayrıca, dağıtım aşamasında karşılaşılan Supabase/IPv6 bağlantı sorunu, bulut servislerinin varsayılan yapılandırmalarının barındırma platformuyla her zaman uyumlu olmayabileceğini; kök neden analizinin (DNS çözümlemesinin IPv6'ya düşmesi) doğru araç ve sorguyla hızlıca teşhis edilebileceğini göstermiştir.

Sonuç olarak proje, bir MVP (Minimum Uygulanabilir Ürün) olarak; gerçek dünya API kısıtlarına uyum sağlayan, maliyet bilinciyle tasarlanmış ve tek bir servis olarak kolayca dağıtılabilen bir yapay zeka çağrı platformu ortaya koymuştur. Gerçek zamanlı çift yönlü ses akışı, kullanıcı başına veri izolasyonu ve gerçek telefon şebekesi (Twilio) entegrasyonunun uçtan uca test edilmesi, ileriye dönük geliştirme alanları olarak belirlenmiştir.

## 8. KAYNAKÇA

- [1] <https://react.dev/>
- [2] <https://vite.dev/>
- [3] <https://www.typescriptlang.org/>
- [4] <https://tailwindcss.com/>
- [5] <https://expressjs.com/>
- [6] <https://node-postgres.com/> (node-postgres / pg)
- [7] <https://huggingface.co/docs/inference-providers/>
- [8] [https://huggingface.co/docs/huggingface.js/inference/README\\_\(@huggingface/inference\)](https://huggingface.co/docs/huggingface.js/inference/README_(@huggingface/inference))
- [9] <https://elevenlabs.io/docs/api-reference/text-to-speech/convert>
- [10] <https://supabase.com/docs/guides/database/connecting-to-postgres>
- [11] [Supabase — IPv4/IPv6 uyumluluk dokümantasyonu](#)
- [12] <https://render.com/docs>
- [13] <https://jwt.io/introduction>
- [14] <https://www.npmjs.com/package/ws>