# Contents

**MinskMir Voice Agent - Детальная Архитектура** **2**
Оглавление . . . 2
Технологический стек . . . 2
Frontend . . . 2
Backend . . . 2
База данных . . . 2
Voice AI & LLM . . . 3
Развертывание . . . 3
Поток данных . . . 3
1. Инициация голосового звонка . . . 3
2. Обработка голосового разговора в ElevenLabs . . . 4
3. Сохранение данных после звонка . . . 5
4. Отображение в Admin Panel . . . 7
5. Real-time уведомления . . . 7
Компоненты системы . . . 8
Frontend Components . . . 8
API эндпоинты . . . 9
ElevenLabs Integration . . . 9
Lead Management . . . 10
Webhooks . . . 11
Knowledge Base . . . 12
База данных . . . 13
Supabase Schema . . . 13
Row Level Security (RLS) . . . 14
Интеграции . . . 15
ElevenLabs Conversational AI . . . 15
Конфигурация ElevenLabs Агента . . . 16
Environment Variables . . . 16
Диаграмма полного потока . . . 17
Технические особенности . . . 19
1. Real-time коммуникация . . . 19
2. Smart Linking алгоритм . . . 19
3. LLM GPT-5 анализ транскриптов . . . 20
4. Безопасность . . . 21
Performance Optimization . . . 21
Мониторинг и логирование . . . 21
Развертывание . . . 22
Vercel Configuration . . . 22
Supabase Migration . . . 22
Roadmap и будущие улучше□□ия . . . 22
RAG система и база знаний . . . 22
Общая концепция . . . 22
Архитектура RAG системы . . . 23
Реализация векторного поиска . . . 24
Интеграция RAG с ElevenLabs Agent . . . 26
API эндпоинт для RAG tool . . . 27
Автоматическое обновление данных . . . 28
Общий принцип . . . 28
Архитектура автообновления . . . 28
Реализация web scraper . . . 29
Обработка изменений и индексация . . . 31
Vercel Cron Job эндпоинт . . . 33

Vercel Cron Configuration . . . . . 34
Мониторинг и алерты . . . . . 34
Database Schema для RAG . . . . . 35
Интеграция с 1C CRM (Real-time sync) . . . . . 36
Безопасность и обработка персональных данных . . . . . 38
Общие принципы . . . . . 38
Категории обрабатываемых персональных данных . . . . . 38
Правовые основания обработки . . . . . 39
Архитектура хранения данных . . . . . 39
Меры защиты данных . . . . . 40
Права субъектов данных . . . . . 42
Срок хранения данных . . . . . 44
Передача данных третьим лицам . . . . . 45
Инциденты с данными . . . . . 46
Согласие пользователя . . . . . 46
Соответствие законодательству Республики Беларусь . . . . . 47
Контактная информация . . . . . 48
Политика конфиденциальности . . . . . 49

# MinskMir Voice Agent - Детальная Архитектура

## Оглавление

1. Технологический стек
2. Поток данных
3. Компоненты системы
4. API эндпоинты
5. База данных
6. Интеграции
7. RAG система и база знаний
8. Автоматическое обновление данных
9. Безопасность и обработка персональных данных

---

## Технологический стек

### Frontend

- **Next.js 13+** - React framework с App Router
- **TypeScript** - Типизированный JavaScript
- **Tailwind CSS** - Utility-first CSS framework
- **React Hooks** - useEffect, useState, useNotifications, useUnreadLeads

### Backend

- **Next.js API Routes** - Serverless функции
- **Node.js** - Runtime environment
- **TypeScript** - Типизация на стороне сервера

### База данных

- **Supabase** - PostgreSQL база данных как сервис

  - Real-time subscriptions
  - Row Level Security (RLS)
  - Автоматические миграции
  - REST API + PostgREST

**Voice AI & LLM**

- **ElevenLabs Conversational AI** - Полный стек голосового AI
  - **Text-to-Speech (TTS)** - Синтез речи (голос AI агента)
  - **Speech-to-Text (STT)** - Распознавание речи пользователя
  - **LLM GPT-5 Integration** - Встроенная интеграция с GPT-5
  - **Conversation Management** - Управление диалогами
  - **WebSocket Support** - Real-time аудио стриминг

**Развертывание**

- **Vercel** - Hosting и CI/CD
- **GitHub** - Version control
- **Vercel Analytics** - Мониторинг производительности

---

## Поток данных

### 1. Инициация голосового звонка

```
                "     "


VoiceWidget.tsx
-

      POST /api/elevenlabs/conversation-token


API: conversation-token.ts
1.       agentId   env
2.       ElevenLabs API
3.        signed JWT token

             { conversationId, token }


VoiceWidget.tsx
-          ElevenLabs SDK
-         WebSocket
```

ElevenLabs Platform
real-time

## 2. Обработка голосового разговора в ElevenLabs

ElevenLabs Platform

1. AUDIO INPUT ( )
   -
   - WebSocket

2. SPEECH-TO-TEXT (STT)
   : ElevenLabs Turbo STT
   - real-time
   - 30+
   - (~300ms)
   Output: " "

3. CONTEXT & KNOWLEDGE BASE
   -
   - (conversation context)
   - Custom tools & functions

4. LLM GPT-5 PROCESSING
   : GPT-5 ( )

   Input:
   -
   - System prompt
   - Knowledge base context
   -

   Processing:
   -
   - KB
   -
   - custom tools ( )

   Output: " 45 120 ²..."

```
5. TEXT-TO-SPEECH (TTS)
        : ElevenLabs TTS v3
   -
   -
   -
   - Streaming audio generation
   Output:



6. AUDIO OUTPUT (          )
   -                      WebSocket
   -                  AI
```

### 3. Сохранение данных после звонка

```
ElevenLabs Platform


              webhook
        POST /api/webhook/voice-lead-enhanced



API: webhook/voice-lead-enhanced.ts

          :
{
  conversation_id: "uuid",
  agent_id: "uuid",
  status: "done",
  transcript: [
    { role: "user", message: "    " },
    { role: "agent", message: "       !" }
  ],
  recording_url: "https://...",
  metadata: {
    phone: "+375291234567",
    name: "       ",
    email: "ivan@example.com"
  }
}



       webhook:
```

```
1.            ElevenLabs
2.                 :
   -   ,     , email
   -           (        )
   -    ,     (        )
3.                  LLM:
   -        intent (      )
   -
   -              (lead scoring)
```

```
Smart Linking (lib/smart-linking.ts)

1.     :                       ?
   SELECT * FROM "Lead" WHERE phone = '+375291234567'

2a.            :
    -             (lastContactDate)
    -           Conversation          Lead

2b.          :
    -          Lead   Supabase
    -            Conversation
```

```
Supabase Database

INSERT INTO "Lead" {
  name: "       ",
  phone: "+375291234567",
  email: "ivan@example.com",
  source: "voice_agent",
  status: "new",
  createdAt: "2024-01-15T10:30:00Z"
}

INSERT INTO "Conversation" {
  conversationId: "elevenlabs-conv-id",
  leadId: lead.id,
  transcript: [...],
  recordingUrl: "https://...",
  status: "done",
  createdAt: "2024-01-15T10:30:00Z"
}
```

```
Real-time
- Supabase Real-time subscription
```

```
- Webhook        200 OK
```

### 4. Отображение в Admin Panel

```
Admin Panel (pages/admin/leads.tsx)
-

      GET /api/admin/enhanced-leads


API: admin/enhanced-leads.ts
1.               Supabase
2. JOIN Lead + Conversation
3.

           :
      {
        leads: [...],
        conversations: [...],
        stats: {
          totalLeads: 150,
          newLeads: 12,
          conversionRate: 0.25
        }
      }


Admin Panel UI
-
-      (    ,    ,      )
-
-
```

### 5. Real-time уведомления

```
AINotificationWidget.tsx
-

      Polling: GET /api/leads/new-leads?since=timestamp


API: leads/new-leads.ts
1.       Supabase
2.          timestamp
3.

           : { leads: [...], count: 3 }


Notification Badge
```

-
- ( )

---

## Компоненты системы

### Frontend Components

**1. VoiceWidget.tsx** **Назначение**: Главный виджет для голосовых звонков

**Функционал**: - Инициация звонка к AI агенту - Управление состоянием звонка (idle, connecting, connected, ended) - Отображение статуса соединения - Встраивание ElevenLabs ConvAI SDK

**Технические детали**:

```
//      flow
1.              "Call"
2.                 API
3.           ElevenLabs Conversation
4.          WebSocket
5.
```

**Используемые API**: - POST `/api/elevenlabs/conversation-token` - получение токена

**2. AINotificationWidget.tsx** **Назначение**: Отображение уведомлений о новых лидах

**Функционал**: - Polling новых лидов каждые 30 секунд - Отображение badge с количеством непрочитанных - Переход к списку лидов при клике

**Технические детали**:

```
// Polling
useEffect(() => {
  const interval = setInterval(() => {
    fetch('/api/leads/new-leads?since=' + lastCheck)
  }, 30000)
}, [])
```

**3. CallButton.tsx** **Назначение**: Переиспользуемая кнопка для звонков

**Props**: - `agentId` - ID ElevenLabs агента - `variant` - стиль кнопки - `onCallStart` - callback при начале звонка - `onCallEnd` - callback при завершении

**4. AudioPlayer.tsx** **Назначение**: Воспроизведение записей разговоров

**Функционал**: - Загрузка аудио по signed URL - Контролы: play, pause, seek - Отображение прогресса и длительности

---

## API эндпоинты

### ElevenLabs Integration

**POST /api/elevenlabs/conversation-token** **Назначение**: Генерация JWT токена для начала разговора

**Request**:

```
{
  "agentId": "string (optional)"
}
```

**Response**:

```
{
  "token": "jwt_token_string"
}
```

**Технология**: - ElevenLabs REST API - Signed JWT с expiration

---

**GET /api/elevenlabs/conversations** **Назначение**: Получение списка всех разговоров из ElevenLabs

**Query params**: - `limit` - количество записей (default: 50) - `cursor` - пагинация

**Response**:

```
{
  "conversations": [
    {
      "conversation_id": "uuid",
      "agent_id": "uuid",
      "status": "done",
      "start_time": "ISO8601",
      "end_time": "ISO8601"
    }
  ],
  "next_cursor": "string"
}
```

---

**POST /api/elevenlabs/signed-url** **Назначение**: Генерация подписанного URL для скачивания аудио

**Request**:

```
{
  "conversationId": "uuid"
}
```

**Response**:

```
{
  "signedUrl": "https://storage.elevenlabs.io/...",
  "expiresAt": "ISO8601"
}
```

**Безопасность**: URL действителен 1 час

---

**Lead Management**

**GET /api/admin/leads** **Назначение**: CRUD операции с лидами

**Methods**: GET, POST, PUT, DELETE

**GET Query params**: - `status` - фильтр по статусу - `source` - фильтр по источнику - `limit`, `offset` - пагинация

**Response**:

```
{
  "leads": [
    {
      "id": "uuid",
      "name": "string",
      "phone": "string",
      "email": "string",
      "source": "voice_agent | crm | manual",
      "status": "new | contacted | qualified | converted",
      "createdAt": "ISO8601",
      "lastContactDate": "ISO8601"
    }
  ],
  "total": 150
}
```

---

**GET /api/admin/enhanced-leads** **Назначение**: Расширенная информация о лидах с разговорами

**Response**:

```
{
  "leads": [...],
  "conversations": {
    "leadId1": [
      {
        "conversationId": "uuid",
        "transcript": [...],
        "recordingUrl": "string",
        "duration": 180,
        "createdAt": "ISO8601"
      }
    ]
  },
  "stats": {
    "totalLeads": 150,
    "newLeadsToday": 12,
    "conversionRate": 0.25
  }
}
```

---

**GET /api/leads/new-leads** **Назначение**: Получение новых лидов для уведомлений

**Query params**: - `since` - timestamp последней проверки

**Response**:

```
{
  "leads": [...],
  "count": 3,
  "lastCheck": "ISO8601"
}
```

---

**Webhooks**

**POST /api/webhook/voice-lead-enhanced** **Назначение**: Обработка завершенных разговоров от ElevenLabs

**Request** (от ElevenLabs):

```
{
  "conversation_id": "uuid",
  "agent_id": "uuid",
  "status": "done | failed | timeout",
  "transcript": [
    {
      "role": "user",
      "message": "string",
      "timestamp": "ISO8601"
    },
    {
      "role": "agent",
      "message": "string",
      "timestamp": "ISO8601"
    }
  ],
  "recording_url": "string",
  "metadata": {
    "phone": "string",
    "name": "string",
    "email": "string",
    "custom_fields": {}
  },
  "analysis": {
    "sentiment": "positive | neutral | negative",
    "intent": "inquiry | complaint | purchase",
    "lead_quality": "hot | warm | cold"
  }
}
```

**Processing**: 1. Валидация webhook подписи 2. Извлечение данных лида из metadata и transcript 3. LLM анализ транскрипта для дополнительных данных 4. Smart linking с существующими лидами 5. Сохранение в Supabase

**Response**: 200 OK

---

**POST /api/webhook/crm-lead-enhanced** **Назначение**: Webhook для интеграции с внешними CRM

**Request**:

```
{
  "lead": {
    "name": "string",
    "phone": "string",
    "email": "string",
    "source": "string",
    "custom_fields": {}
  }
}
```

**Processing**: - Создание или обновление лида в Supabase - Дедупликация по телефону/email - Уведомление через real-time

---

**Knowledge Base**

**GET /api/knowledge-base/list** **Назначение**: Список документов в базе знаний

**Response**:

```
{
  "documents": [
    {
      "id": "uuid",
      "name": "string",
      "type": "text | url | file",
      "content": "string",
      "agentIds": ["uuid"],
      "createdAt": "ISO8601"
    }
  ]
}
```

---

**POST /api/knowledge-base/create-text** **Назначение**: Добавление текстового документа

**Request**:

```
{
  "name": "             ",
  "content": "         ..."
}
```

---

**POST /api/knowledge-base/create-url** **Назначение**: Добавление URL в базу знаний

**Request**:

```
{
  "name": "   -  ",
  "url": "https://example.com/price-list"
}
```

**Processing**: - Загрузка контента по URL - Парсинг HTML/PDF - Индексация для RAG

---

**POST /api/knowledge-base/assign-to-agent** **Назначение**: Привязка документа к агенту

**Request**:

```
{
  "documentId": "uuid",
  "agentId": "uuid"
}
```

---

## База данных

### Supabase Schema

### Таблица: Lead

```
CREATE TABLE "Lead" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT,
  phone TEXT UNIQUE,
  email TEXT,
  source TEXT, -- 'voice_agent', 'crm', 'manual', 'website'
  status TEXT DEFAULT 'new', -- 'new', 'contacted', 'qualified', 'converted', 'lost'
  notes TEXT,
  interest TEXT, --
  budget TEXT,
  timeline TEXT, --
  "lastContactDate" TIMESTAMP,
  "createdAt" TIMESTAMP DEFAULT NOW(),
  "updatedAt" TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_lead_phone ON "Lead"(phone);
CREATE INDEX idx_lead_status ON "Lead"(status);
CREATE INDEX idx_lead_created ON "Lead"("createdAt" DESC);
```

---

### Таблица: Conversation

```
CREATE TABLE "Conversation" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  "conversationId" TEXT UNIQUE, -- ElevenLabs conversation ID
```

```
  "leadId" UUID REFERENCES "Lead"(id) ON DELETE CASCADE,
  "agentId" TEXT, -- ElevenLabs agent ID
  transcript JSONB, --
  "recordingUrl" TEXT,
  duration INTEGER, --
  status TEXT, -- 'done', 'failed', 'timeout'
  sentiment TEXT, -- 'positive', 'neutral', 'negative'
  intent TEXT, --
  "leadQuality" TEXT, -- 'hot', 'warm', 'cold'
  "createdAt" TIMESTAMP DEFAULT NOW(),
  "updatedAt" TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_conversation_lead ON "Conversation"("leadId");
CREATE INDEX idx_conversation_elevenlabs ON "Conversation"("conversationId");
CREATE INDEX idx_conversation_created ON "Conversation"("createdAt" DESC);
```

---

### Таблица: KnowledgeBase

```
CREATE TABLE "KnowledgeBase" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  type TEXT, -- 'text', 'url', 'file'
  content TEXT,
  url TEXT,
  "fileUrl" TEXT,
  metadata JSONB, --
  "agentIds" TEXT[], --     agent IDs
  "createdAt" TIMESTAMP DEFAULT NOW(),
  "updatedAt" TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_kb_agent ON "KnowledgeBase" USING GIN("agentIds");
```

---

### Таблица: QuizSubmission

```
CREATE TABLE "QuizSubmission" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  "leadId" UUID REFERENCES "Lead"(id),
  answers JSONB, --
  score INTEGER, --
  "createdAt" TIMESTAMP DEFAULT NOW()
);
```

---

### Row Level Security (RLS)

```
--       RLS
ALTER TABLE "Lead" ENABLE ROW LEVEL SECURITY;
ALTER TABLE "Conversation" ENABLE ROW LEVEL SECURITY;
```

```
--       (       authenticated users)
CREATE POLICY "Allow read for authenticated users"
ON "Lead" FOR SELECT
TO authenticated
USING (true);

CREATE POLICY "Allow insert for service role"
ON "Lead" FOR INSERT
TO service_role
WITH CHECK (true);
```

---

## Интеграции

### ElevenLabs Conversational AI

**Базовый URL**: `https://api.elevenlabs.io/v1`

**Используемые эндпоинты**:

1. **Создание токена для разговора**

```
POST /convai/conversation/get_signed_url
Headers:
  xi-api-key: YOUR_API_KEY
Body:
  {
    "agent_id": "uuid"
  }
Response:
  {
    "signed_url": "wss://...",
    "conversation_id": "uuid"
  }
```

2. **Получение списка разговоров**

```
GET /convai/conversations
Headers:
  xi-api-key: YOUR_API_KEY
Query:
  ?agent_id=uuid&limit=50&cursor=string
```

3. **Получение деталей разговора**

```
GET /convai/conversations/{conversation_id}
Headers:
  xi-api-key: YOUR_API_KEY
Response:
  {
    "conversation_id": "uuid",
    "transcript": [...],
    "recording_url": "string",
    "metadata": {}
  }
```

4. **Webhook настройки** (в ElevenLabs Dashboard)

```
URL: https://your-domain.com/api/webhook/voice-lead-enhanced
Events: conversation.completed
Secret: your_webhook_secret (       )
```

---

### Конфигурация ElevenLabs Агента

**В Dashboard ElevenLabs настраивается**:

1. **Voice Settings**
   - Выбор голоса (мужской/женский, язык)
   - Настройки эмоциональности
   - Скорость речи
2. **LLM Configuration**
   - Модель: GPT-5
   - System prompt (роль агента, инструкции)
   - Temperature (креативность ответов)
   - Max tokens
3. **Knowledge Base**
   - Документы с информацией о продукте
   - FAQs
   - Скрипты продаж
4. **Custom Tools** (функции, которые агент может вызывать)
   - Поиск в базе данных квартир
   - Расчет ипотеки
   - Бронирование просмотра
5. **Webhooks**
   - URL для отправки данных после звонка
   - События для триггера webhook

---

### Environment Variables

```
# ElevenLabs
ELEVENLABS_API_KEY=sk_...
ELEVENLABS_AGENT_ID=agent_...
ELEVENLABS_WEBHOOK_SECRET=whsec_...

# Supabase
NEXT_PUBLIC_SUPABASE_URL=https://xxx.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ...
SUPABASE_SERVICE_ROLE_KEY=eyJ...

# Optional:    LLM
OPENAI_API_KEY=sk-...

# Vercel
VERCEL_URL=your-domain.vercel.app
```

---

## Диаграмма полного потока

```
    website

        1.      "     "

    VoiceWidget.tsx
-       conversation token

        2. POST /api/elevenlabs/conversation-token

 Next.js API Route
-      ElevenLabs API
-      signed JWT

        3.       token

 VoiceWidget.tsx
-         ElevenLabs SDK
-       WebSocket

        4. WebSocket

         ElevenLabs Platform

  Audio       STT        LLM       TTS
  Input                    GPT-5




        Knowledge Base
  -
  -    -
  - FAQs



        5. Real-time

                AI
```

6.

ElevenLabs webhook
POST /api/webhook/voice-lead-enhanced

Payload:
- conversation_id
- transcript ( )
- recording_url
- metadata ( , )

7. Webhook

voice-lead-enhanced.ts

1.
2. LLM transcript:
   - ,
   - interest, budget
   - Lead quality scoring
3. Smart linking ( )
4. Supabase:
   - Lead ( )
   - Conversation ( )

8. Database insert

Supabase PostgreSQL

Tables:
- Lead ( )
- Conversation ( )

9. Real-time notification

AINotificationWidget.tsx
-
- badge

10. Admin Panel

pages/admin/leads.tsx
-

-
-

---

## Технические особенности

### 1. Real-time коммуникация

**WebSocket для голосовых звонков**: - Бидирекциональный аудио стрим - Низкая латентность (~200-500ms end-to-end) - Автоматическое переподключение при обрыве

**Supabase Real-time**: - Подписка на INSERT в таблице Lead - Автоматическое обновление UI без перезагрузки

---

### 2. Smart Linking алгоритм

```
// lib/smart-linking.ts
async function linkConversationToLead(conversationData) {
  const phone = extractPhone(conversationData.metadata);

  // 1.
  let lead = await supabase
    .from('Lead')
    .select('*')
    .eq('phone', phone)
    .single();

  if (!lead) {
    // 2.       email (     )
    const email = extractEmail(conversationData.metadata);
    if (email) {
      lead = await supabase
        .from('Lead')
        .select('*')
        .eq('email', email)
        .single();
    }
  }

  if (!lead) {
    // 3.
    lead = await supabase
      .from('Lead')
      .insert({
        name: extractName(conversationData.transcript),
        phone,
        email,
        source: 'voice_agent',
        status: 'new'
      })
      .select()
```

```
      .single();
  } else {
    // 4.
    await supabase
      .from('Lead')
      .update({ lastContactDate: new Date() })
      .eq('id', lead.id);
  }

  // 5.       conversation
  await supabase
    .from('Conversation')
    .insert({
      conversationId: conversationData.conversation_id,
      leadId: lead.id,
      transcript: conversationData.transcript,
      recordingUrl: conversationData.recording_url,
      status: conversationData.status
    });

  return lead;
}
```

---

### 3. LLM GPT-5 анализ транскриптов

```
//
async function analyzeTranscript(transcript) {
  const prompt = `
                              :
    1.
    2.
    3. Email (         )
    4.          (       ,    )
    5.
    6.
    7.          (hot/warm/cold)

          :
    ${JSON.stringify(transcript)}

               JSON.
  `;

  const response = await openai.chat.completions.create({
    model: 'gpt-5',
    messages: [{ role: 'user', content: prompt }],
    response_format: { type: 'json_object' }
  });

  return JSON.parse(response.choices[0].message.content);
}
```

---

### 4. Безопасность

**Webhook валидация**:

```
//             ElevenLabs
function verifyWebhookSignature(payload, signature, secret) {
  const hmac = crypto.createHmac('sha256', secret);
  const digest = hmac.update(payload).digest('hex');
  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(digest)
  );
}
```

**API защита**: - Supabase Row Level Security (RLS) - API keys в environment variables - CORS настройки для веб-виджета

---

## Performance Optimization

1. **Кэширование**:
   - Next.js автоматический static generation
   - Supabase query caching
   - CDN для аудио файлов (через signed URLs)
2. **Lazy Loading**:
   - Компоненты загружаются по требованию
   - Аудио файлы не загружаются до клика
3. **Database индексы**:
   - На часто используемых полях (phone, status, createdAt)
   - GIN индекс для JSONB полей

---

## Мониторинг и логирование

```
// lib/api-logger.ts
export function logAPICall(endpoint: string, data: any) {
  console.log({
    timestamp: new Date().toISOString(),
    endpoint,
    data,
    environment: process.env.NODE_ENV
  });

  //              (      )
  // await sendToSentry(...)
}
```

---

## Развертывание

### Vercel Configuration

```
// vercel.json
{
  "env": {
    "ELEVENLABS_API_KEY": "@elevenlabs-api-key",
    "SUPABASE_SERVICE_ROLE_KEY": "@supabase-service-key"
  },
  "regions": ["iad1"],
  "functions": {
    "api/**/*.ts": {
      "maxDuration": 30
    }
  }
}
```

### Supabase Migration

```
#
npx supabase db push

# Seed
npx supabase db seed
```

---

## Roadmap и будущие улучше□□ия

1. **AI Analytics Dashboard**
   - Визуализация метрик разговоров
   - A/B тестирование промптов агента
   - Анализ conversion funnel
2. **Multi-language support**
   - Автоопределение языка пользователя
   - Переключение голоса агента
3. **CRM интеграция**
   - Синхронизация с 1C CRM
   - Двусторонняя синхронизация данных
4. **Advanced lead scoring**
   - ML модель для предсказания conversion
   - Автоматическая приоритизация лидов

---

## RAG система и база знаний

### Общая концепция

Система использует **RAG (Retrieval-Augmented Generation)** для предоставления AI-агенту актуальной информации о квартирах, ценах и услугах. Все данные о недвижимости хранятся в векторной базе данных и автоматически индексируются для быстрого семантического поиска.

**Архитектура RAG системы**

```
                1  CRM
 bir.by                          (PDF/DOCX)


                     real-time


1.
   - Web scraping (Puppeteer/Cheerio)
   - API        1
   -             (PDF/DOCX)


2.
   -       HTML
   -
   -


3. Chunking (              )
   -      chunk: 512
   - Overlap: 50
   -


4.          (Embeddings)
       : text-embedding-3-large (OpenAI)
          : 3072
      :     /
```

```
    Pinecone / Supabase pgvector

        :
    -       embeddings (3072 dim)
    -            chunk
    -       :
      * apartment_id
      * address
      * price
      * area
      * rooms
      * floor
      * source (website/crm/document)
      * last_updated

        :
    - HNSW (Hierarchical Navigable Small World)
    -     : cosine similarity
```

### Реализация векторного поиска

```typescript
// lib/rag/vector-search.ts
import { OpenAI } from 'openai';
import { createClient } from '@supabase/supabase-js';

const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });
const supabase = createClient(
  process.env.SUPABASE_URL,
  process.env.SUPABASE_KEY
);

//      embedding
async function createEmbedding(text: string): Promise<number[]> {
  const response = await openai.embeddings.create({
    model: 'text-embedding-3-large',
    input: text,
    dimensions: 3072
  });

  return response.data[0].embedding;
}

//
async function searchApartments(query: string, filters?: {
  minPrice?: number;
  maxPrice?: number;
  rooms?: number;
  district?: string;
}) {
  // 1.      embedding
```

```
  const queryEmbedding = await createEmbedding(query);

  // 2.                  Supabase
  const { data: results, error } = await supabase.rpc('match_apartments', {
    query_embedding: queryEmbedding,
    match_threshold: 0.75, //
    match_count: 10, //   -10
    filter_min_price: filters?.minPrice,
    filter_max_price: filters?.maxPrice,
    filter_rooms: filters?.rooms,
    filter_district: filters?.district
  });

  if (error) throw error;

  return results.map(result => ({
    apartmentId: result.apartment_id,
    address: result.address,
    price: result.price,
    area: result.area,
    rooms: result.rooms,
    floor: result.floor,
    description: result.content,
    similarity: result.similarity,
    lastUpdated: result.last_updated
  }));
}

// PostgreSQL
/*
CREATE OR REPLACE FUNCTION match_apartments(
  query_embedding vector(3072),
  match_threshold float,
  match_count int,
  filter_min_price int DEFAULT NULL,
  filter_max_price int DEFAULT NULL,
  filter_rooms int DEFAULT NULL,
  filter_district text DEFAULT NULL
)
RETURNS TABLE (
  apartment_id uuid,
  address text,
  price int,
  area float,
  rooms int,
  floor int,
  content text,
  similarity float,
  last_updated timestamp
)
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN QUERY
```

```
  SELECT
    a.id,
    a.address,
    a.price,
    a.area,
    a.rooms,
    a.floor,
    a.description,
    1 - (a.embedding <=> query_embedding) as similarity,
    a.updated_at
  FROM apartments a
  WHERE
    (filter_min_price IS NULL OR a.price >= filter_min_price)
    AND (filter_max_price IS NULL OR a.price <= filter_max_price)
    AND (filter_rooms IS NULL OR a.rooms = filter_rooms)
    AND (filter_district IS NULL OR a.district = filter_district)
    AND 1 - (a.embedding <=> query_embedding) > match_threshold
  ORDER BY a.embedding <=> query_embedding
  LIMIT match_count;
END;
$$;
*/
```

### Интеграция RAG с ElevenLabs Agent

```
// lib/elevenlabs/rag-tool.ts
// Custom tool    ElevenLabs

export const apartmentSearchTool = {
  name: 'search_apartments',
  description: '                                    ',
  parameters: {
    type: 'object',
    properties: {
      query: {
        type: 'string',
        description: '                 (    , "           2     ")'
      },
      minPrice: {
        type: 'number',
        description: '           USD (      )'
      },
      maxPrice: {
        type: 'number',
        description: '            USD (      )'
      },
      rooms: {
        type: 'number',
        description: '          (      )'
      },
      district: {
        type: 'string',
        description: '        (      )'
      }
```

```
    },
    required: ['query']
  },
  handler: async (params: {
    query: string;
    minPrice?: number;
    maxPrice?: number;
    rooms?: number;
    district?: string;
  }) => {
    //
    const results = await searchApartments(params.query, {
      minPrice: params.minPrice,
      maxPrice: params.maxPrice,
      rooms: params.rooms,
      district: params.district
    });

    //
    if (results.length === 0) {
      return {
        success: true,
        message: '       ,                    .                    .',
        apartments: []
      };
    }

    return {
      success: true,
      message: `     ${results.length}            `,
      apartments: results.map(apt => ({
        address: apt.address,
        price: `${apt.price} USD`,
        area: `${apt.area}  ²`,
        rooms: apt.rooms,
        floor: apt.floor,
        description: apt.description,
        relevance: Math.round(apt.similarity * 100) + '%'
      }))
    };
  }
};

//        tool   ElevenLabs
//   Dashboard ElevenLabs -> Agent Settings -> Custom Tools:
//       webhook endpoint: POST /api/tools/search-apartments
```

### API эндпоинт для RAG tool

```
// pages/api/tools/search-apartments.ts
import type { NextApiRequest, NextApiResponse } from 'next';
import { searchApartments } from '@/lib/rag/vector-search';

export default async function handler(
```

```
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method not allowed' });
  }

  //         ElevenLabs
  const signature = req.headers['x-elevenlabs-signature'];
  if (!verifyElevenLabsSignature(signature, req.body)) {
    return res.status(401).json({ error: 'Invalid signature' });
  }

  const { query, minPrice, maxPrice, rooms, district } = req.body;

  try {
    const results = await searchApartments(query, {
      minPrice,
      maxPrice,
      rooms,
      district
    });

    res.status(200).json({
      success: true,
      results: results.slice(0, 5) //   -5
    });
  } catch (error) {
    console.error('RAG search error:', error);
    res.status(500).json({
      success: false,
      error: '             '
    });
  }
}
```

---

## Автоматическое обновление данных

### Общий принцип

Система автоматически обновляет информацию о квартирах с сайта **bir.by каждый час** с помощью web scraping и синхронизирует данные с 1C CRM в **real-time** через webhooks.

### Архитектура автообновления

```
                Vercel Cron Jobs

       : 0 * * * * (      )
      : /api/cron/scrape-apartments
```

Web Scraper (Puppeteer)

1. bir.by/apartments
2. :
   - , , ,
   - , ,
3. ( / / )

Diff Engine ( )

-
- delta ( )
-

1. embeddings /
2. Upsert Supabase pgvector
3. (last_updated)

ElevenLabs Knowledge Base

- API
-

### Реализация web scraper

```typescript
// lib/scraper/bir-scraper.ts
import puppeteer from 'puppeteer';
import * as cheerio from 'cheerio';

interface ApartmentData {
  externalId: string;  // ID
  address: string;
  price: number;
  area: number;
  rooms: number;
  floor: number;
  totalFloors: number;
  district: string;
  description: string;
  photos: string[];
```

```
  url: string;
  scrapedAt: Date;
}

export async function scrapeBirApartments(): Promise<ApartmentData[]> {
  const browser = await puppeteer.launch({
    headless: true,
    args: ['--no-sandbox', '--disable-setuid-sandbox']
  });

  const page = await browser.newPage();
  const apartments: ApartmentData[] = [];

  try {
    //
    await page.goto('https://bir.by/apartments', {
      waitUntil: 'networkidle2',
      timeout: 30000
    });

    //
    await page.waitForSelector('.apartment-card', { timeout: 10000 });

    //      HTML
    const html = await page.content();
    const $ = cheerio.load(html);

    //
    $('.apartment-card').each((index, element) => {
      const $card = $(element);

      const apartment: ApartmentData = {
        externalId: $card.attr('data-id') || `apt-${index}`,
        address: $card.find('.address').text().trim(),
        price: parsePrice($card.find('.price').text()),
        area: parseFloat($card.find('.area').text().replace(/[^\d.]/g, '')),
        rooms: parseInt($card.find('.rooms').text().replace(/\D/g, '')) || 0,
        floor: parseInt($card.find('.floor').text().split('/')[0]?.replace(/\D/g, '')) || 0,
        totalFloors: parseInt($card.find('.floor').text().split('/')[1]?.replace(/\D/g, '')) || 0,
        district: $card.find('.district').text().trim(),
        description: $card.find('.description').text().trim(),
        photos: $card.find('.photo img').map((i, img) => $(img).attr('src')).get(),
        url: 'https://bir.by' + $card.find('a').attr('href'),
        scrapedAt: new Date()
      };

      apartments.push(apartment);
    });

    //      :            "          "
    const hasNextPage = await page.$('.pagination .next');
    if (hasNextPage) {
      await page.click('.pagination .next');
      await page.waitForNavigation({ waitUntil: 'networkidle2' });
```

```
      //                        (           )
    }

  } catch (error) {
    console.error('Scraping error:', error);
    throw error;
  } finally {
    await browser.close();
  }

  return apartments;
}

function parsePrice(priceText: string): number {
  //                    "$120,000"    "120 000 $"
  return parseInt(priceText.replace(/[^\d]/g, '')) || 0;
}
```

### Обработка изменений и индексация

```
// lib/scraper/indexer.ts
import { createClient } from '@supabase/supabase-js';
import { createEmbedding } from '@/lib/rag/vector-search';
import type { ApartmentData } from './bir-scraper';

const supabase = createClient(
  process.env.SUPABASE_URL!,
  process.env.SUPABASE_SERVICE_ROLE_KEY!
);

export async function indexApartments(apartments: ApartmentData[]) {
  const results = {
    added: 0,
    updated: 0,
    unchanged: 0,
    errors: 0
  };

  for (const apt of apartments) {
    try {
      //
      const { data: existing } = await supabase
        .from('apartments')
        .select('id, price, description, updated_at')
        .eq('external_id', apt.externalId)
        .single();

      //      embedding
      const textForEmbedding = `
           : ${apt.address}
           : ${apt.district}
          : ${apt.price} USD
            : ${apt.area}  ²
           : ${apt.rooms}
```

```
        : ${apt.floor}/${apt.totalFloors}
          : ${apt.description}
      `.trim();

      const embedding = await createEmbedding(textForEmbedding);

      if (!existing) {
        //          -
        await supabase.from('apartments').insert({
          external_id: apt.externalId,
          address: apt.address,
          price: apt.price,
          area: apt.area,
          rooms: apt.rooms,
          floor: apt.floor,
          total_floors: apt.totalFloors,
          district: apt.district,
          description: apt.description,
          photos: apt.photos,
          url: apt.url,
          embedding: embedding,
          source: 'website',
          scraped_at: apt.scrapedAt
        });

        results.added++;
      } else {
        //                -
        const hasChanges =
          existing.price !== apt.price ||
          existing.description !== apt.description;

        if (hasChanges) {
          //
          await supabase
            .from('apartments')
            .update({
              price: apt.price,
              area: apt.area,
              description: apt.description,
              photos: apt.photos,
              embedding: embedding,
              updated_at: new Date().toISOString(),
              scraped_at: apt.scrapedAt
            })
            .eq('id', existing.id);

          results.updated++;
        } else {
          results.unchanged++;
        }
      }
    } catch (error) {
      console.error(`Error indexing apartment ${apt.externalId}:`, error);
```

```
      results.errors++;
    }
  }

  return results;
}

//          ,
export async function cleanupOldListings(currentExternalIds: string[]) {
  const { data: removed } = await supabase
    .from('apartments')
    .update({ status: 'sold_or_removed', updated_at: new Date().toISOString() })
    .eq('source', 'website')
    .not('external_id', 'in', `(${currentExternalIds.join(',')})`)
    .eq('status', 'active')
    .select('id');

  return removed?.length || 0;
}
```

### Vercel Cron Job эндпоинт

```
// pages/api/cron/scrape-apartments.ts
import type { NextApiRequest, NextApiResponse } from 'next';
import { scrapeBirApartments } from '@/lib/scraper/bir-scraper';
import { indexApartments, cleanupOldListings } from '@/lib/scraper/indexer';

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  //      Vercel Cron
  const authHeader = req.headers.authorization;
  if (authHeader !== `Bearer ${process.env.CRON_SECRET}`) {
    return res.status(401).json({ error: 'Unauthorized' });
  }

  try {
    console.log('[CRON] Starting apartment scraping...');
    const startTime = Date.now();

    // 1.
    const apartments = await scrapeBirApartments();
    console.log(`[CRON] Scraped ${apartments.length} apartments`);

    // 2.
    const indexResults = await indexApartments(apartments);
    console.log('[CRON] Indexing results:', indexResults);

    // 3.           ,
    const externalIds = apartments.map(apt => apt.externalId);
    const removedCount = await cleanupOldListings(externalIds);
    console.log(`[CRON] Marked ${removedCount} listings as removed`);
```

```
    const duration = Date.now() - startTime;

    res.status(200).json({
      success: true,
      duration: `${duration}ms`,
      stats: {
        scraped: apartments.length,
        added: indexResults.added,
        updated: indexResults.updated,
        unchanged: indexResults.unchanged,
        removed: removedCount,
        errors: indexResults.errors
      },
      timestamp: new Date().toISOString()
    });
  } catch (error) {
    console.error('[CRON] Scraping failed:', error);
    res.status(500).json({
      success: false,
      error: error.message,
      timestamp: new Date().toISOString()
    });
  }
}
```

### Vercel Cron Configuration

```
// vercel.json
{
  "crons": [
    {
      "path": "/api/cron/scrape-apartments",
      "schedule": "0 * * * *"
    }
  ]
}
```

**Расписание**: `0 * * * *` = каждый час в 0 минут (00:00, 01:00, 02:00, … 23:00)

### Мониторинг и алерты

```
// lib/monitoring/scraper-monitor.ts
import { createClient } from '@supabase/supabase-js';

//
export async function logScraperRun(stats: {
  scraped: number;
  added: number;
  updated: number;
  removed: number;
  errors: number;
  duration: number;
}) {
  const supabase = createClient(
```

```
    process.env.SUPABASE_URL!,
    process.env.SUPABASE_SERVICE_ROLE_KEY!
  );

  await supabase.from('scraper_logs').insert({
    scraped_count: stats.scraped,
    added_count: stats.added,
    updated_count: stats.updated,
    removed_count: stats.removed,
    error_count: stats.errors,
    duration_ms: stats.duration,
    timestamp: new Date().toISOString()
  });

  //
  if (stats.errors > 10 || stats.scraped === 0) {
    await sendAlert({
      type: 'scraper_error',
      message: `Scraper issues detected. Errors: ${stats.errors}, Scraped: ${stats.scraped}`,
      stats
    });
  }
}

async function sendAlert(alert: any) {
  //        Slack/Telegram/Email
  await fetch(process.env.SLACK_WEBHOOK_URL!, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      text: `  [BIR.BY Scraper Alert]\n${alert.message}`,
      attachments: [{
        fields: Object.entries(alert.stats).map(([key, value]) => ({
          title: key,
          value: String(value),
          short: true
        }))
      }]
    })
  });
}
```

### Database Schema для RAG

```
--
CREATE TABLE apartments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  external_id TEXT UNIQUE NOT NULL,
  address TEXT NOT NULL,
  price INTEGER NOT NULL,
  area FLOAT NOT NULL,
  rooms INTEGER NOT NULL,
  floor INTEGER,
  total_floors INTEGER,
```

```
  district TEXT,
  description TEXT,
  photos TEXT[],
  url TEXT,
  embedding vector(3072), --    pgvector
  source TEXT DEFAULT 'website', -- 'website', 'crm', 'manual'
  status TEXT DEFAULT 'active', -- 'active', 'sold_or_removed'
  scraped_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

--
CREATE INDEX idx_apartments_external ON apartments(external_id);
CREATE INDEX idx_apartments_status ON apartments(status);
CREATE INDEX idx_apartments_price ON apartments(price);
CREATE INDEX idx_apartments_rooms ON apartments(rooms);
CREATE INDEX idx_apartments_district ON apartments(district);

-- HNSW                    (pgvector)
CREATE INDEX idx_apartments_embedding ON apartments
USING hnsw (embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64);

--
CREATE TABLE scraper_logs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  scraped_count INTEGER,
  added_count INTEGER,
  updated_count INTEGER,
  removed_count INTEGER,
  error_count INTEGER,
  duration_ms INTEGER,
  timestamp TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_scraper_logs_timestamp ON scraper_logs(timestamp DESC);
```

### Интеграция с 1C CRM (Real-time sync)

```
// pages/api/webhook/1c-apartment-update.ts
// Webhook                  1  CRM

import type { NextApiRequest, NextApiResponse } from 'next';
import { createEmbedding } from '@/lib/rag/vector-search';
import { createClient } from '@supabase/supabase-js';

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method not allowed' });
  }
```

```
//      API       1
const apiKey = req.headers['x-api-key'];
if (apiKey !== process.env.CRM_API_KEY) {
  return res.status(401).json({ error: 'Unauthorized' });
}

const { action, apartment } = req.body;
// action: 'create' | 'update' | 'delete'

const supabase = createClient(
  process.env.SUPABASE_URL!,
  process.env.SUPABASE_SERVICE_ROLE_KEY!
);

try {
  if (action === 'create' || action === 'update') {
    //      embedding
    const textForEmbedding = `
         : ${apartment.address}
        : ${apartment.price} USD
          : ${apartment.area}  ²
         : ${apartment.rooms}
          : ${apartment.description}
    `.trim();

    const embedding = await createEmbedding(textForEmbedding);

    // Upsert
    await supabase.from('apartments').upsert({
      external_id: apartment.crm_id,
      address: apartment.address,
      price: apartment.price,
      area: apartment.area,
      rooms: apartment.rooms,
      floor: apartment.floor,
      total_floors: apartment.total_floors,
      description: apartment.description,
      embedding: embedding,
      source: 'crm',
      status: apartment.status,
      updated_at: new Date().toISOString()
    }, { onConflict: 'external_id' });

  } else if (action === 'delete') {
    //
    await supabase
      .from('apartments')
      .update({ status: 'sold_or_removed' })
      .eq('external_id', apartment.crm_id);
  }

  res.status(200).json({ success: true });
} catch (error) {
```

```
    console.error('1C webhook error:', error);
    res.status(500).json({ success: false, error: error.message });
  }
}
```

---

## Безопасность и обработка персональных данных

### Общие принципы

Система MinskMir Voice Agent обрабатывает персональные данные пользователей в строгом соответствии с законодательством Республики Беларусь:

**Нормативно-правовая база**: - Закон Республики Беларусь от 7 мая 2021 г. № 99-З «О защите персональных данных» - Указ Президента Республики Беларусь от 7 ноября 2022 г. № 383 «О мерах по совершенствованию отношений в области обработки персональных данных» - Постановление Совета Министров Республики Беларусь от 30 августа 2022 г. № 568 «Об утверждении Положения об информационной безопасности персональных данных»

**Основной сервер развертывания**: Республика Беларусь (дата-центр на территории РБ)

**Статус оператора**: ООО "МинскМир" зарегистрировано в качестве оператора персональных данных в установленном порядке

---

### Категории обрабатываемых персональных данных

В соответствии со статьей 5 Закона РБ «О защите персональных данных», обрабатываются следующие категории персональных данных:

**1. Общедоступные персональные данные (статья 7 Закона) Из голосовых разговоров**: - Фамилия, имя, отчество - Номер телефона (с согласия субъекта) - Адрес электронной почты (с согласия субъекта) - Сведения о потребностях в недвижимости (открыто предоставленные субъектом)

**Из веб-взаимодействий**: - IP-адрес (обезличенный после 90 дней) - Информация о браузере и операционной системе (техническая информация) - Cookie-файлы (с согласия пользователя в соответствии с требованиями законодательства)

**2. Специальные персональные данные Аудио и видеозаписи** (статья 8 Закона): - Аудиозаписи телефонных разговоров (обрабатываются только с письменного согласия субъекта) - Текстовые транскрипты разговоров - Время и дата обращения - Метаданные разговора

**Из интеграции с 1C CRM**: - История обращений и покупок - Статус клиента - Дополнительные контактные данные (адрес проживания при наличии согласия)

**3. Обезличенные данные** В соответствии со статьей 6 Закона, после истечения срока хранения персональные данные подлежат обезличиванию для статистических целей: - Агрегированная статистика обращений - Анонимизированные данные для улучшения AI-модели - Общие метрики конверсии

---

**Правовые основания обработки**

В соответствии со статьей 12 Закона РБ «О защите персональных данных»:

**1. Согласие субъекта персональных данных (статья 13 Закона) Форма согласия**: Письменное или в форме электронного документа

**Содержание согласия**: - Фамилия, имя, отчество, адрес субъекта персональных данных - Наименование и адрес оператора персональных данных - Цели обработки персональных данных - Перечень действий с персональными данными - Срок действия согласия - Способ отзыва согласия

**Реализация**:

```
//
interface Consent {
  subjectName: string;           //
  subjectAddress?: string;       //     (          )
  operatorName: string;          //    "     "
  operatorAddress: string;       //
  purposes: string[];            //
  actions: string[];             //
  validUntil: Date;              //
  consentDate: Date;             //
  consentMethod: 'verbal' | 'written' | 'electronic'; //
  audioRecordingUrl?: string;    //                 (      )
  withdrawalMethod: string;      //
}
```

**При первом голосовом контакте** AI-агент озвучивает: > “Здравствуйте! Я голосовой помощник компании МинскМир. > В соответствии с законодательством Республики Беларусь о защите персональных данных, сообщаю: > Этот разговор записывается. Ваши персональные данные (ФИО, номер телефона, сведения о потребностях) будут обработаны компанией ООО ‘МинскМир’ в целях предоставления консультации и оказания услуг. > Вы имеете право на доступ к своим данным, их исправление и удаление. Подробности на сайте bir.by/privacy. > Продолжая разговор, вы даете согласие на обработку ваших персональных данных. > Если вы не согласны, пожалуйста, прекратите разговор сейчас.”

**2. Исполнение договора (часть 1 статьи 12 Закона)** Обработка необходима для: - Заключения и исполнения договора купли-продажи недвижимости - Предоставления консультационных услуг - Ведения переговоров по заключению договора

**3. Обработка в рамках законодательства РБ**

- Налоговый учет (хранение 5 лет в соответствии с Налоговым кодексом РБ)
- Бухгалтерская отчетность
- Статистическая отчетность (обезличенные данные)

---

**Архитектура хранения данных**

**Локация серверов**

```
                  (Primary Region)


    Supabase Instance (EU/BY Region)
    - PostgreSQL Database
    -
    -
    -



    Application Server (Vercel Edge - BY)
    - Next.js API Routes
    -       webhook
    - Admin Panel




        Third-Party Services (      )


    ElevenLabs (USA/EU)
    -                (24-72   )
    - STT/TTS
    -
    - Data Processing Agreement (DPA)



    OpenAI (USA)
    -
    -
    - Data Processing Agreement (DPA)
```

---

**Меры защиты данных**

**1. Шифрование  В транзите**:

```
//    API            HTTPS/TLS 1.3
// WebSocket            WSS (WebSocket Secure)

//
const wsConfig = {
  url: 'wss://api.elevenlabs.io/...',
  rejectUnauthorized: true,
  minVersion: 'TLSv1.3'
};
```

**В состоянии покоя**: - Supabase использует AES-256 шифрование для данных в базе - Аудио файлы хранятся в зашифрованном S3-совместимом хранилище - Резервные копии шифруются автоматически

### 2. Контроль доступа  Row Level Security (RLS) в Supabase:

```
--
CREATE POLICY "Managers can view leads"
ON "Lead" FOR SELECT
TO authenticated
USING (
  auth.jwt() ->> 'role' = 'manager' OR
  auth.jwt() ->> 'role' = 'admin'
);

--
CREATE POLICY "Only admins can delete"
ON "Lead" FOR DELETE
TO authenticated
USING (auth.jwt() ->> 'role' = 'admin');

--
CREATE POLICY "Log all lead changes"
ON "Lead" FOR ALL
TO authenticated
USING (true)
WITH CHECK (
  log_data_change(
    TG_TABLE_NAME,
    TG_OP,
    row_to_json(NEW),
    auth.uid()
  )
);
```

**API ключи и токены**:

```
//      API          90
//               IP
// Rate limiting

// Middleware
export function requireAuth(req: NextApiRequest) {
  const token = req.headers.authorization;
  if (!token || !verifyJWT(token)) {
    throw new UnauthorizedError();
  }

  //      IP whitelist    admin endpoints
  if (req.url?.startsWith('/api/admin/')) {
    if (!isIPWhitelisted(req.socket.remoteAddress)) {
      logSecurityEvent('UNAUTHORIZED_ADMIN_ACCESS', req);
      throw new ForbiddenError();
    }
  }
```

```
}
```

**3. Аудит и логирование  Журнал доступа к персональным данным**:

```
CREATE TABLE "DataAccessLog" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  userId UUID NOT NULL,
  action TEXT NOT NULL, -- 'read', 'update', 'delete', 'export'
  tableName TEXT NOT NULL,
  recordId UUID,
  ipAddress INET,
  userAgent TEXT,
  timestamp TIMESTAMP DEFAULT NOW(),
  dataSnapshot JSONB --
);

--
CREATE INDEX idx_access_log_user ON "DataAccessLog"(userId);
CREATE INDEX idx_access_log_time ON "DataAccessLog"(timestamp DESC);
CREATE INDEX idx_access_log_action ON "DataAccessLog"(action);
```

**Автоматическое логирование**:

```
//
async function logDataAccess(params: {
  userId: string;
  action: 'read' | 'update' | 'delete' | 'export';
  tableName: string;
  recordId: string;
  ipAddress: string;
}) {
  await supabase.from('DataAccessLog').insert({
    ...params,
    timestamp: new Date().toISOString()
  });
}
```

---

**Права субъектов данных**

**1. Право на доступ**  Пользователь может запросить копию всех своих данных.

**API эндпоинт**:

```
// GET /api/gdpr/export-my-data
//       JSON/PDF

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  const { phone } = req.query;

  //               ( SMS  )
  await verifySMSCode(phone, req.body.code);

  //
  const userData = {
```

```
    lead: await getLeadByPhone(phone),
    conversations: await getConversationsByPhone(phone),
    recordings: await getRecordingURLs(phone), //       signed URLs
    quizSubmissions: await getQuizSubmissions(phone),
    exportDate: new Date().toISOString()
  };

  //
  await logDataAccess({
    userId: userData.lead.id,
    action: 'export',
    tableName: 'Lead',
    recordId: userData.lead.id,
    ipAddress: req.socket.remoteAddress
  });

  res.json(userData);
}
```

**2. Право на исправление** Пользователь может исправить неточные данные через admin panel или API.

**3. Право на удаление ("право быть забытым") API эндпоинт**:

```
// DELETE /api/gdpr/delete-my-data
export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  const { phone, verificationCode } = req.body;

  //
  await verifySMSCode(phone, verificationCode);

  const lead = await getLeadByPhone(phone);

  //          (soft delete)
  await supabase.from('Lead').update({
    name: '[DELETED]',
    phone: null,
    email: null,
    deletedAt: new Date().toISOString(),
    notes: '[                    ]'
  }).eq('id', lead.id);

  //
  await deleteRecordings(lead.id);
  await anonymizeTranscripts(lead.id);

  //
  await logDataAccess({
    userId: lead.id,
    action: 'delete',
    tableName: 'Lead',
    recordId: lead.id,
    ipAddress: req.socket.remoteAddress
  });
```

```
  res.json({ success: true, message: '          ' });
}
```

**4. Право на ограничение обработки** Пользователь может запретить использование данных для маркетинга.

```
ALTER TABLE "Lead" ADD COLUMN "processingRestrictions" JSONB DEFAULT '{
  "marketing": false,
  "analytics": false,
  "aiTraining": false
}'::jsonb;
```

---

**Срок хранения данных**

| Тип данных | Срок хранения | Основание |
|---|---|---|
| Активные лиды | До конвертации + 1 год | Законные интересы |
| Конвертированные клиенты | 5 лет с даты покупки | Налоговое законодательство РБ |
| Неактивные лиды | 2 года с последнего контакта | Законные интересы |
| Аудио записи | 90 дней | Контроль качества |
| Транскрипты | 2 года | Обучение AI, споры |
| Логи доступа | 3 года | Кибербезопасность |
| Анонимизированная аналитика | Бессрочно | Статистика |

**Автоматическое удаление**:

```
// Cron job:
export async function cleanupOldData() {
  const now = new Date();

  //              90
  const ninetyDaysAgo = subDays(now, 90);
  await deleteOldRecordings(ninetyDaysAgo);

  //                    2
  const twoYearsAgo = subYears(now, 2);
  await anonymizeInactiveLeads(twoYearsAgo);

  //              3
  const threeYearsAgo = subYears(now, 3);
  await deleteOldLogs(threeYearsAgo);

  console.log('Data cleanup completed', { timestamp: now });
}
```

---

**Передача данных третьим лицам**

**1. ElevenLabs (обработчик) Data Processing Agreement (DPA)**: - Данные обрабатываются только для предоставления сервиса - Не используются для обучения моделей ElevenLabs без согласия - Аудио хранится максимум 72 часа на серверах ElevenLabs - Шифрование данных в транзите и покое

**Конфигурация**:

```
//
const elevenLabsConfig = {
  retention_policy: {
    audio_files: '72h', //            72
    transcripts: '7d',  //             7
    metadata: '30d'     //       30
  },
  data_usage: {
    training: false,    //
    analytics: true     //
  }
};
```

**2. OpenAI (обработчик) Настройки конфиденциальности**:

```
// API       opt-out
const openaiConfig = {
  model: 'gpt-5',
  //
  // https://openai.com/enterprise-privacy
  headers: {
    'OpenAI-Organization': 'org-xxx',
    'X-Data-Usage': 'no-training' //
  }
};
```

**3. Интеграция с 1C CRM Двусторонняя синхронизация**:

```
// Webhook             1
POST https://1c-server.company.by/hs/leads/webhook
Headers:
  X-API-Key: secret_key
  Content-Type: application/json
Body:
  {
    "lead_id": "uuid",
    "name": "encrypted_name",
    "phone_hash": "sha256_hash", //
    "status": "new",
    "source": "voice_agent"
  }

//
function encryptForCRM(data: Lead) {
  return {
    ...data,
```

```
    name: encrypt(data.name, CRM_PUBLIC_KEY),
    phone: hashPhone(data.phone), //
    email: data.email ? encrypt(data.email, CRM_PUBLIC_KEY) : null
  };
}
```

---

**Инциденты с данными**

**Процедура реагирования  1. Обнаружение инцидента** (в течение 1 часа):

```
//
async function detectSecurityIncident() {
  //
  const failedAttempts = await checkFailedLogins();

  //
  const anomalies = await detectAnomalies();

  //
  const leaks = await checkDataLeaks();

  if (failedAttempts.isCritical || anomalies.length > 0 || leaks) {
    await triggerIncidentResponse();
  }
}
```

**2.  Оценка масштаба** (в течение 6 часов):  - Какие данные затронуты - Сколько пользователей - Тип инцидента (утечка, несанкционированный доступ, потеря)

**3. Уведомление** (в течение 72 часов):  - Уполномоченный орган по защите данных РБ - Затронутые пользователи (если высокий риск) - Руководство компании

**4. Устранение** (немедленно):

```
//
async function containIncident(incidentId: string) {
  //              API
  await revokeAPIKeys(incidentId);

  //                IP
  await blockIPs(incidentId);

  //
  await rotateCredentials();

  //
  await createIncidentSnapshot(incidentId);
}
```

---

**Согласие пользователя**

**Механизм получения согласия  При первом звонке**:

```
// AI Agent        ( system prompt)
const systemPrompt = `
  -                  MinskMir.

   :
          :

"       !                        .
                                                .
          ,                                    .
                              .          ?"

                          .
`;

//
function recordConsent(conversationId: string) {
  return {
    conversation_id: conversationId,
    consent: {
      given: true,
      timestamp: new Date().toISOString(),
      type: 'verbal',
      recording_url: 'link_to_consent_part'
    }
  };
}
```

**На веб-сайте**:

```
// Cookie banner
<CookieConsent
  location="bottom"
  buttonText="    "
  declineButtonText="     "
  cookieName="minsk_mir_gdpr_consent"
>
          cookie
         . <Link href="/privacy">               </Link>
</CookieConsent>
```

---

**Соответствие законодательству Республики Беларусь**

**Закон РБ от 7 мая 2021 г. № 99-З «О защите персональных данных» Обязанности оператора (статья 17)**:

☐ **Обеспечение безопасности персональных данных** (статья 17): - Разработка и утверждение локальных нормативных правовых актов - Назначение лица, ответственного за организацию обработки персональных данных - Применение правовых, организационных и технических мер защиты - Контроль за принимаемыми мерами по обеспечению безопасности

☐ **Получение согласия субъекта** (статья 13): - Получение письменного согласия в установленной форме - Информирование о целях, способах и сроках обработки - Указание

на право отозвать согласие

☐ **Информирование субъекта** (статья 19): - Предоставление информации о наличии персональных данных - Предоставление копии персональных данных (бесплатно 1 раз в год) - Уведомление об изменениях, удалении, уточнении данных

☐ **Обеспечение прав субъектов** (статья 20): - Право на доступ к своим персональным данным - Право на уточнение, изменение, удаление данных - Право на отзыв согласия

☐ **Уведомление об инцидентах** (статья 17): - Уведомление Оперативно-аналитического центра при Президенте РБ - Уведомление субъектов данных при утечке - Срок уведомления: незамедлительно, но не позднее 72 часов

**Указ Президента РБ от 7 ноября 2022 г. № 383** ☐ **Регистрация оператора**: - ООО "МинскМир" включено в реестр операторов персональных данных - Уведомление Оперативно-аналитического центра при Президенте РБ - Регистрационный номер: [указывается после регистрации]

☐ **Локализация данных**: - Основная база данных размещена на территории Республики Беларусь - Резервное копирование на территории РБ - Трансграничная передача данных осуществляется в соответствии со статьей 22 Закона

**Постановление Совета Министров РБ от 30 августа 2022 г. № 568** ☐ **Положение об информационной безопасности**: - Внедрена система защиты персональных данных (СЗПД) - Использование сертифицированных средств криптографической защиты информации (СКЗИ) - Аттестация информационной системы по требованиям безопасности - Регулярное обновление мер безопасности

☐ **Организационные меры**: - Разработана политика обработки персональных данных - Проведено обучение сотрудников - Назначен ответственный за организацию обработки персональных данных - Установлен порядок доступа к персональным данным

☐ **Технические меры**: - Шифрование персональных данных (ГОСТ 34.12-2018, ГОСТ 34.13-2018) - Контроль доступа и аутентификация - Журналирование и аудит действий с персональными данными - Защита от несанкционированного доступа

**Трансграничная передача данных (статья 22 Закона)** Передача персональных данных в иностранные государства осуществляется при наличии: - Письменного согласия субъекта персональных данных - Уведомления Оперативно-аналитического центра при Президенте РБ - Договоров с принимающей стороной о защите данных (DPA)

**Текущие трансграничные передачи**: 1. **ElevenLabs (США/ЕС)** - временное хранение аудио для обработки (72 часа) - Заключен Data Processing Agreement - Получено согласие субъектов - Уведомлен ОАЦ при Президенте РБ

2. **OpenAI (США)** - обработка транскриптов для извлечения данных
   - Заключен Data Processing Agreement
   - Данные НЕ используются для обучения
   - Уведомлен ОАЦ при Президенте РБ

---

**Контактная информация**

**Оператор персональных данных Полное наименование**: Общество с ограниченной ответственностью "МинскМир" **Сокращенное наименование**: ООО "МинскМир" **УНП**: [указывается УНП] **Юридический адрес**: 220000, Республика Беларусь, г. Минск, ул.

[указывается адрес] **Почтовый адрес**: 220000, Республика Беларусь, г. Минск, ул. [указывается адрес]

**Лицо, ответственное за организацию обработки персональных данных ФИО**: [Фамилия Имя Отчество] **Должность**: Ответственный за организацию обработки персональных данных **Email**: privacy@bir.by **Телефон**: +375 (XX) XXX-XX-XX **Приемные часы**: понедельник-пятница, 9:00-18:00

**Обращения субъектов персональных данных По вопросам обработки персональных данных**: - Email: privacy@bir.by - Почтовый адрес: 220000, г. Минск, ул. [адрес], ООО "МинскМир" - Телефон: +375 (XX) XXX-XX-XX

**Запросы на доступ к данным**: data-access@bir.by **Запросы на уточнение, изменение, удаление данных**: data-update@bir.by **Отзыв согласия**: consent-withdrawal@bir.by **Жалобы на действия оператора**: complaints@bir.by

**Сроки рассмотрения обращений (в соответствии со статьей 19 Закона)**

- Предоставление информации о наличии персональных данных: **не позднее 15 дней** с даты получения обращения
- Предоставление копии персональных данных (первый раз бесплатно): **не позднее 30 дней**
- Уточнение, изменение персональных данных: **не позднее 10 дней**
- Удаление персональных данных: **не позднее 3 дней** (если нет законных оснований для хранения)
- Отзыв согласия: вступает в силу **со дня получения** оператором

**Уполномоченный орган Оперативно-аналитический центр при Президенте Республики Беларусь** - Адрес: 220013, г. Минск, ул. Я. Коласа, 37 - Телефон: +375 (17) 374-91-51 - Email: info@oac.gov.by - Сайт: https://oac.gov.by

**Для жалоб на действия оператора**: обращения направляются в письменной форме или в форме электронного документа

---

**Политика конфиденциальности**

Полная версия политики конфиденциальности доступна: - На сайте: https://bir.by/privacy - В приложении к договору - По запросу в офисе компании

**Последнее обновление**: Январь 2025

---

*Раздел обновлен: 2025-01-15*