Voice Processing App - Architecture & Design Document

1. System Overview

Core Features

1. **Text-to-Speech (TTS)** - Convert written text to spoken audio
2. **Voice Library & Upload** - Access voice library or upload custom voice samples
3. **Accent Modification** - Change speech accent dynamically
4. **Language Dubbing** - Convert speech to different languages
5. **Voice Editing** - Edit individual words in generated speech
6. **Noise Reduction** - Remove background noise from audio
7. **Background Music Generation** - Create ambient music tracks
8. **Sound Effects** - Generate various sound effects
9. **Speech-to-Text (STT)** - Convert spoken audio to written text

Technology Stack Recommendations

- **Frontend Web**: React.js/Next.js with TypeScript
- **Mobile**: React Native or Flutter
- **Backend**: Node.js/Express or Python/FastAPI
- **Database**: PostgreSQL (metadata) + MongoDB (audio metadata)
- **File Storage**: AWS S3 or Google Cloud Storage
- **AI/ML Services**: OpenAI Whisper, ElevenLabs, Azure Cognitive Services
- **Real-time Processing**: WebRTC, Socket.io
- **Audio Processing**: FFmpeg, WebAudio API

---

2. High-Level Architecture

System Architecture Diagram

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Web Client    │    │  Mobile Client  │    │   Admin Panel   │
│   (React.js)    │    │ (React Native)  │    │   (React.js)    │
└─────────────────┘    └─────────────────┘    └─────────────────┘
          │                      │                      │
          └──────────────────────┼──────────────────────┘
                                 │
                    ┌─────────────────────────────────────────────┐
                    │               API Gateway                   │
                    │           (Rate Limiting,                   │
                    │       Authentication, Routing)              │
                    └─────────────────────────────────────────────┘
                                 │
          ┌──────────────────────┼──────────────────────┐
          │                      │                      │
    ┌────▼───┐ ┌────▼────┐ ┌────▼────┐ ┌────▼────┐ ┌────▼────┐
    │ Auth   │ │ Voice   │ │ Audio   │ │ ML/AI   │ │ File    │
    │Service │ │Processing│ │Enhancement│ │Services │ │Storage │
    │        │ │ Service │ │ Service │ │         │ │Service  │
    └────────┘ └─────────┘ └─────────┘ └─────────┘ └─────────┘
                   │            │           │           │
           ┌───────▼─────┐ ┌────▼─────┐ ┌──▼───┐ ┌─────▼────┐
           │ Voice/Audio │ │ Metadata │ │Queue │ │ CDN      │
           │ Storage     │ │ Database │ │(Redis)│ │         │
           │ (S3/GCS)    │ │(PostgreSQL│ │      │ │         │
           │             │ │/MongoDB) │ │      │ │         │
           └─────────────┘ └──────────┘ └──────┘ └─────────┘
```

Core Components

2.1 Client Layer

- **Web Application**: Progressive Web App with offline capabilities
- **Mobile Application**: Cross-platform with native audio processing
- **Admin Dashboard**: Content management and analytics

2.2 API Gateway

- Request routing and load balancing
- Authentication and authorization
- Rate limiting and throttling
- API versioning and documentation

2.3 Microservices Architecture

- **Authentication Service**: User management, JWT tokens
- **Voice Processing Service**: Core TTS/STT functionality
- **Audio Enhancement Service**: Noise reduction, effects
- **ML/AI Services**: Voice cloning, accent modification
- **File Storage Service**: Audio file management

2.4 Data Layer

- **Primary Database**: User accounts, projects, metadata
- **File Storage**: Raw audio files, processed outputs
- **Cache Layer**: Frequently accessed data, session management
- **Message Queue**: Asynchronous processing tasks

---

3. Low-Level Design

3.1 Database Schema

Users Table

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    subscription_tier VARCHAR(50) DEFAULT 'free',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

Projects Table

```sql
CREATE TABLE projects (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    settings JSONB,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

Voice Profiles Table

```sql
CREATE TABLE voice_profiles (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    name VARCHAR(255) NOT NULL,
    voice_type VARCHAR(50), -- 'preset', 'uploaded', 'cloned'
    voice_data JSONB, -- voice parameters, model references
    sample_file_url VARCHAR(500),
    is_public BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Audio Files Table

```sql
CREATE TABLE audio_files (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID REFERENCES projects(id),
    file_name VARCHAR(255) NOT NULL,
    file_url VARCHAR(500) NOT NULL,
    file_type VARCHAR(50), -- 'tts_output', 'uploaded', 'processed'
    metadata JSONB,
    processing_status VARCHAR(50) DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT NOW()
);
```

3.2 API Design

Authentication Endpoints

```
POST /api/v1/auth/register
POST /api/v1/auth/login
POST /api/v1/auth/refresh
DELETE /api/v1/auth/logout
```

Voice Processing Endpoints

```
POST /api/v1/tts/convert
GET /api/v1/voices/library
POST /api/v1/voices/upload
POST /api/v1/voices/clone
PUT /api/v1/voices/{id}/accent
PUT /api/v1/voices/{id}/language
```

Audio Enhancement Endpoints

```
POST /api/v1/audio/denoise
POST /api/v1/audio/edit-word
POST /api/v1/audio/background-music
POST /api/v1/audio/sound-effects
```

Speech-to-Text Endpoints

```
POST /api/v1/stt/transcribe
GET /api/v1/stt/status/{job_id}
```

3.3 Service Implementations

Text-to-Speech Service

```javascript
class TTSService {
    async convertTextToSpeech(params) {
        const {
            text,
            voiceId,
            language,
            accent,
            speed,
            pitch
        } = params;

        // Validate input
        if (!text || text.length > 10000) {
            throw new ValidationError('Invalid text input');
        }

        // Process through AI service
        const audioBuffer = await this.aiProvider.synthesize({
            text,
            voice: voiceId,
            language,
            accent,
            speed,
            pitch
        });

        // Store audio file
        const fileUrl = await this.storageService.uploadAudio(audioBuffer);

        // Save metadata
        await this.database.saveAudioFile({
            fileUrl,
            metadata: params,
            type: 'tts_output'
        });

        return { fileUrl, duration: audioBuffer.duration };
    }
}
```

Voice Cloning Service

```javascript
class VoiceCloning {
    async cloneVoice(audioSample, userId) {
        // Validate audio sample
        const validation = await this.validateAudioSample(audioSample);
        if (!validation.isValid) {
            throw new Error(validation.error);
        }

        // Extract voice features
        const voiceFeatures = await this.extractVoiceFeatures(audioSample);

        // Train voice model
        const modelId = await this.trainVoiceModel(voiceFeatures);

        // Save voice profile
        const voiceProfile = await this.database.saveVoiceProfile({
            userId,
            modelId,
            features: voiceFeatures,
            type: 'cloned'
        });

        return voiceProfile;
    }
}
```

Audio Enhancement Service

javascript

```javascript
class AudioEnhancement {
    async removeNoise(audioFileUrl) {
        // Download audio file
        const audioBuffer = await this.downloadAudio(audioFileUrl);

        // Apply noise reduction
        const denoisedBuffer = await this.applyNoiseReduction(audioBuffer);

        // Upload processed file
        const processedUrl = await this.uploadProcessedAudio(denoisedBuffer);

        return { processedUrl };
    }

    async editWord(audioFileUrl, wordPosition, newWord, voiceId) {
        // Extract audio segments
        const segments = await this.segmentAudio(audioFileUrl);

        // Generate new word audio
        const newWordAudio = await this.ttsService.synthesizeWord(newWord, voiceId);

        // Replace segment
        const editedAudio = await this.replaceSegment(segments, wordPosition, newWordA

        // Upload result
        const resultUrl = await this.uploadProcessedAudio(editedAudio);

        return { resultUrl };
    }
}
```

3.4 Real-time Processing Architecture

WebSocket Connection Management

```javascript
class AudioProcessor {
    constructor() {
        this.activeConnections = new Map();
        this.processingQueue = new Queue();
    }

    handleWebSocketConnection(socket, userId) {
        this.activeConnections.set(userId, socket);

        socket.on('audio-chunk', async (chunk) => {
            await this.processRealTimeAudio(chunk, userId);
        });

        socket.on('start-recording', () => {
            this.initializeRecording(userId);
        });

        socket.on('stop-recording', async () => {
            const result = await this.finalizeRecording(userId);
            socket.emit('recording-complete', result);
        });
    }
}
```

3.5 Mobile-Specific Considerations

React Native Audio Module

```javascript
// Custom native module for audio processing
const AudioProcessorModule = {
    startRecording: (config) => {
        return NativeModules.AudioProcessor.startRecording(config);
    },

    stopRecording: () => {
        return NativeModules.AudioProcessor.stopRecording();
    },

    playAudio: (url) => {
        return NativeModules.AudioProcessor.playAudio(url);
    },

    processAudioRealTime: (audioData) => {
        return NativeModules.AudioProcessor.processAudioRealTime(audioData);
    }
};
```

---

4. Security Considerations

4.1 Authentication & Authorization

- JWT-based authentication with refresh tokens
- Role-based access control (RBAC)
- OAuth2 integration for third-party login
- Multi-factor authentication for premium users

4.2 Data Protection

- End-to-end encryption for sensitive audio data
- PII data anonymization
- Secure file upload with virus scanning
- Rate limiting to prevent abuse

4.3 Audio File Security

- Signed URLs for file access
- Automatic file expiration
- Content validation and sanitization
- Watermarking for premium content

5. Scalability & Performance

5.1 Horizontal Scaling

- Microservices deployed in containers (Docker/Kubernetes)
- Auto-scaling based on CPU/memory usage
- Load balancing across multiple instances
- Database read replicas for improved performance

5.2 Caching Strategy

- Redis for session management and frequent queries
- CDN for static audio files
- Application-level caching for voice models
- Browser caching for web assets

5.3 Performance Optimization

- Lazy loading for large audio files
- Audio streaming instead of full downloads
- Progressive enhancement for mobile devices
- Background processing for heavy tasks

---

6. Monitoring & Analytics

6.1 System Monitoring

- Application performance monitoring (APM)
- Real-time error tracking and alerting
- Infrastructure monitoring (CPU, memory, disk)
- API response time and success rate tracking

6.2 Business Analytics

- User engagement metrics
- Feature usage analytics
- Conversion funnel analysis
- A/B testing framework

---

7. Deployment Architecture

7.1 CI/CD Pipeline

```yaml
# Example GitHub Actions workflow
name: Deploy Voice App
on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: npm test

  build:
    runs-on: ubuntu-latest
    steps:
      - name: Build Docker images
        run: docker build -t voice-app .

  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to production
        run: kubectl apply -f k8s/
```

7.2 Infrastructure as Code

- Terraform for cloud resource provisioning
- Kubernetes for container orchestration
- Helm charts for application deployment
- Environment-specific configurations

This architecture provides a robust, scalable foundation for your voice processing application with clear separation of concerns and room for future enhancements.