Flutter + LiveKit + bitHuman Integration Guide

This comprehensive guide shows how to integrate Flutter with LiveKit and bitHuman Cloud Essence to applications with AI-powered avatars.

🏗️ Architecture Overview

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
|   Flutter App   |    |   LiveKit Room  |    |  Python Agent   |
|                 |    |                 |    |                 |
|  ┌───────────┐  |    |  ┌───────────┐  |    |  ┌───────────┐  |
|  |   Video   |  |◄──►|  | Real-time |  |◄──►|  | bitHuman  |  |
|  |   View    |  |    |  | Streaming |  |    |  |  Avatar   |  |
|  └───────────┘  |    |  └───────────┘  |    |  └───────────┘  |
|                 |    |                 |    |                 |
|  ┌───────────┐  |    |  ┌───────────┐  |    |  ┌───────────┐  |
|  |   Audio   |  |◄──►|  |   Audio   |  |◄──►|  |  OpenAI   |  |
|  |  Capture  |  |    |  |  Routing  |  |    |  |    API    |  |
|  └───────────┘  |    |  └───────────┘  |    |  └───────────┘  |
└─────────────────┘    └─────────────────┘    └─────────────────┘
```

🔑 Why a Token Server is required (Production)

- LiveKit requires a JWT to join rooms. Creating this JWT needs your LiveKit API key and secret, whic (Flutter).
- Provide a tiny server endpoint `/token` that mints short-lived tokens with room grants and identi
- For development you can hardcode a token in Flutter; for production use the token endpoint.

Minimal endpoint (Python/Flask):

```python
@app.route('/token', methods=['POST'])
def create_token():
    data = request.get_json() or {}
    room = data.get('room', 'flutter-avatar-room')
    identity = data.get('participant', 'Flutter User')
    at = api.AccessToken(LIVEKIT_API_KEY, LIVEKIT_API_SECRET, identity=identity)
    at.add_grant(api.VideoGrant(room_join=True, room=room))
    at.ttl = timedelta(hours=1)
    return jsonify({ 'token': at.to_jwt(), 'server_url': LIVEKIT_URL })
```

Component Responsibilities

- **Flutter App**: Cross-platform UI, camera/microphone capture, video rendering
- **LiveKit Room**: Real-time media routing, participant management, signaling
- **Python Agent**: AI conversation processing, avatar rendering, media coordination

🚀 Quick Start

Prerequisites

- Flutter SDK 3.0+
- Python 3.11+
- bitHuman API Secret
- LiveKit Cloud account
- OpenAI API Key

Complete Setup Guide

Step 1: Project Structure

```
mkdir flutter-bithuman-avatar
cd flutter-bithuman-avatar
mkdir -p backend frontend/lib
```

Step 2: Backend Configuration

```
cd backend

# Create requirements.txt
cat > requirements.txt << EOF
livekit-agents==0.6.0
livekit-plugins-openai==0.6.0
livekit-plugins-silero==0.6.0
bithuman==0.1.0
flask==3.0.0
python-dotenv==1.0.0
EOF

# Create environment configuration
cat > .env.example << EOF
# bitHuman API Configuration
BITHUMAN_API_SECRET=sk_bh_your_secret_here
BITHUMAN_AVATAR_ID=A33NZN6384

# OpenAI API Configuration
OPENAI_API_KEY=sk-proj_your_key_here

# LiveKit Configuration
LIVEKIT_API_KEY=APIyour_key
LIVEKIT_API_SECRET=your_secret
LIVEKIT_URL=wss://your-project.livekit.cloud
EOF

# Setup Python environment
cp .env.example .env
# Edit .env with your actual API keys
python3 -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate
pip install -r requirements.txt
```

Step 3: Frontend Configuration

```
cd ../frontend

# Initialize Flutter project
flutter create . --org com.bithuman.avatar

# Update pubspec.yaml with LiveKit dependencies
cat > pubspec.yaml << EOF
name: flutter_bithuman_avatar
description: Flutter app with LiveKit and bitHuman AI avatar integration

version: 1.0.0+1

environment:
  sdk: '>=3.0.0 <4.0.0'
  flutter: ">=3.10.0"

dependencies:
  flutter:
    sdk: flutter
  livekit_components: 1.2.2+hotfix.1
  livekit_client: ^2.5.3
  provider: ^6.1.1
  http: ^1.1.0
  flutter_dotenv: ^5.1.0

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^3.0.0

flutter:
  uses-material-design: true
  assets:
    - .env
EOF

flutter pub get
```

Step 4: Run the System

```
# Terminal 1: Start Backend
cd backend
source .venv/bin/activate
python token_server.py &
python agent.py dev

# Terminal 2: Start Frontend
cd frontend
flutter run -d chrome --web-port 8080
```

Flutter CLI Installation (macOS)

```
brew install --cask flutter
echo 'export PATH="$PATH:/Applications/flutter/bin"' >> ~/.zprofile
```

```
source ~/.zprofile
flutter --version
flutter doctor
flutter config --enable-web
```

Environment Configuration

The Flutter app supports environment variables for configuration:

```
# Required: LiveKit server URL
export LIVEKIT_SERVER_URL="wss://your-project.livekit.cloud"

# Token configuration (choose one)
export LIVEKIT_TOKEN_ENDPOINT="http://localhost:3000/token"  # Recommended
# OR
export LIVEKIT_TOKEN="your-jwt-token-here"  # Testing only

# Optional: Room configuration
export LIVEKIT_ROOM_NAME="flutter-avatar-room"
export LIVEKIT_PARTICIPANT_NAME="Flutter User"
```

See `frontend/CONFIG.md` for detailed configuration options.

Platform Support

This Flutter app supports multiple platforms:

- **Web**: Run in browser (fastest to test)
- **Android**: Mobile app with camera/microphone permissions
- **iOS**: Mobile app with camera/microphone permissions

Quick test on web:

```
cd frontend
flutter run -d chrome
```

Check available devices:

```
flutter devices
```

Troubleshooting

If you encounter shader compilation errors on macOS:

```
# Clean and rebuild
flutter clean
flutter pub get
flutter run -d chrome --web-port 8080

# Alternative: Use different port
```

```
flutter run -d chrome --web-port 3000
```

If Flutter doctor shows issues:

```
# Install missing dependencies
brew install cocoapods
flutter doctor --android-licenses
```

📱 Flutter Implementation

Core Flutter Code

The Flutter app uses LiveKit Components for production-ready video calling UI:

1. pubspec.yaml

```
name: flutter_bithuman_avatar
description: Flutter app with LiveKit and bitHuman AI avatar integration

version: 1.0.0+1

environment:
  sdk: '>=3.0.0 <4.0.0'
  flutter: ">=3.10.0"

dependencies:
  flutter:
    sdk: flutter

  # LiveKit Components (production-ready UI)
  livekit_components: 1.2.2+hotfix.1
  livekit_client: ^2.5.3

  # State management
  provider: ^6.1.1

  # HTTP requests
  http: ^1.1.0

  # Environment variables
  flutter_dotenv: ^5.1.0

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^3.0.0

flutter:
  uses-material-design: true
  assets:
    - .env
```

2. main.dart (Complete Implementation)

```
import 'package:flutter/material.dart';
import 'package:livekit_client/livekit_client.dart' as lk;
import 'package:livekit_components/livekit_components.dart';
import 'package:logging/logging.dart';

import 'config/livekit_config.dart';

// Create logger instance
final _logger = Logger('BitHumanFlutter');
import 'dart:convert';
import 'dart:math';

void main() {
  // Initialize logger (show info level and above)
  Logger.root.level = Level.INFO;
  Logger.root.onRecord.listen((record) {
    print('${record.level.name}: ${record.time}: ${record.message}');
  });

  runApp(const BitHumanFlutterApp());
}

class BitHumanFlutterApp extends StatelessWidget {
  const BitHumanFlutterApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'bitHuman Flutter Integration',
      theme: LiveKitTheme().buildThemeData(context),
      themeMode: ThemeMode.dark,
      home: const ConnectionScreen(),
      debugShowCheckedModeBanner: false,
    );
  }
}

/// Connection screen - handles token generation and room joining
class ConnectionScreen extends StatefulWidget {
  const ConnectionScreen({super.key});

  @override
  State<ConnectionScreen> createState() => _ConnectionScreenState();
}

class _ConnectionScreenState extends State<ConnectionScreen> {
  final Logger _logger = Logger('ConnectionScreen');
  bool _isConnecting = false;
  String? _error;

  @override
  void initState() {
    super.initState();
    // Auto-connect on startup
    WidgetsBinding.instance.addPostFrameCallback((_) {
      _connect();
    });
  }

  Future<void> _connect() async {
    setState(() {
      _isConnecting = true;
      _error = null;
    });

    try {
      // Get configuration
      final serverUrl = LiveKitConfig.serverUrl;
      final roomName = LiveKitConfig.roomName;
      final participantName = LiveKitConfig.participantName;

      _logger.info('Connecting to room: $roomName as $participantName');
      _logger.info('Server: $serverUrl');

      // Get token from token server
      final token = await LiveKitConfig.getToken();
      _logger.info('Token obtained successfully');

      if (!mounted) return;

      // Navigate to video room using LiveKit Components
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(
          builder: (_) => VideoRoomScreen(
            url: serverUrl,
            token: token,
            roomName: roomName,
          ),
        ),
      );
    } catch (e) {
      _logger.severe('Connection failed: $e');
      setState(() {
        _error = e.toString();
        _isConnecting = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFF1a1a1a),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const CircularProgressIndicator(
              valueColor: AlwaysStoppedAnimation<Color>(Colors.blue),
            ),
            const SizedBox(height: 20),
            Text(
              _isConnecting ? 'Connecting to AI Avatar...' : 'Connection Failed',
              style: const TextStyle(
                color: Colors.white70,
                fontSize: 18,
              ),
            ),
            if (_error != null) ...[
              const SizedBox(height: 16),
              Text(
                _error!,
                style: const TextStyle(
                  color: Colors.red,
                  fontSize: 14,
                ),
                textAlign: TextAlign.center,
              ),
              const SizedBox(height: 16),
              ElevatedButton(
                onPressed: _connect,
                child: const Text('Retry'),
              ),
            ],
          ],
        ),
      ),
    );
  }

}
    const tokenEndpoint = 'http://localhost:3000/token';

    try {
      final response = await http.post(
        Uri.parse(tokenEndpoint),
        headers: {'Content-Type': 'application/json'},
        body: jsonEncode({
          'room': roomName,
          'participant': participantName,
        }),
      );

      if (response.statusCode == 200) {
        return jsonDecode(response.body);
      } else {
        throw Exception('Token server error: ${response.statusCode}');
      }
    } catch (e) {
      throw Exception('Failed to get token: $e');
    }
  }
}

/// Video room screen using LiveKit Components for full-screen AI Avatar display
class VideoRoomScreen extends StatefulWidget {
  final String url;
  final String token;
  final String roomName;

  const VideoRoomScreen({
    super.key,
    required this.url,
    required this.token,
    required this.roomName,
  });

  @override
  State<VideoRoomScreen> createState() => _VideoRoomScreenState();
```

```
}

class _VideoRoomScreenState extends State<VideoRoomScreen> {
  @override
  Widget build(BuildContext context) {
    // Use LiveKit Components' LivekitRoom widget
    return LivekitRoom(
      roomContext: RoomContext(
        url: widget.url,
        token: widget.token,
        connect: true,
        roomOptions: lk.RoomOptions(
          adaptiveStream: true,
          dynacast: true,
          // Enable microphone by default as per LiveKit docs
          defaultAudioPublishOptions: lk.AudioPublishOptions(
            dtx: true,
          ),
          defaultVideoPublishOptions: lk.VideoPublishOptions(
            simulcast: true,
          ),
        ),
      ),
      builder: (context, roomCtx) {
        // Enable microphone by default as per LiveKit docs
        WidgetsBinding.instance.addPostFrameCallback((_) {
          try {
            roomCtx.room.localParticipant?.setMicrophoneEnabled(true);
            _logger.info('🎤 Microphone enabled by default');
          } catch (error) {
            _logger.warning('Could not enable microphone, error: $error');
          }
        });

        return Scaffold(
          appBar: AppBar(
            title: Text('Room: ${widget.roomName}'),
            backgroundColor: const Color(0xFF1a1a1a),
            actions: [
              // Connection status indicator
              Padding(
                padding: const EdgeInsets.all(16),
                child: Row(
                  children: [
                    Icon(
                      roomCtx.room.connectionState == lk.ConnectionState.connecte
                          ? Icons.circle
                          : Icons.circle_outlined,
                      color: roomCtx.room.connectionState == lk.ConnectionState.
                          ? Colors.green
                          : Colors.red,
                      size: 12,
                    ),
                    const SizedBox(width: 8),
                    Text(
                      roomCtx.room.connectionState.toString().split('.').last,
                      style: const TextStyle(fontSize: 12),
                    ),
                  ],
                ),
              ),
            ],
          ),
          backgroundColor: const Color(0xFF1a1a1a),
          body: Stack(
            children: [
              // Full-screen video for AI Avatar (remote participants only)
              Positioned.fill(
                child: Container(
                  color: Colors.black,
                  child: _VideoDisplayWidget(roomCtx: roomCtx),
                ),
              ),

              // Audio handling - separate from video to prevent re-rendering
              Positioned.fill(
                child: _AudioHandlerWidget(roomCtx: roomCtx),
              ),

              // Loading indicator overlay (only show when no remote video)
              Positioned.fill(
                child: _LoadingOverlay(),
              ),

              // Control bar at the bottom (floating over video)
              Positioned(
                left: 0,
                right: 0,
                bottom: 0,
                child: Container(
                  decoration: BoxDecoration(
                    gradient: LinearGradient(
                      begin: Alignment.topCenter,
                      end: Alignment.bottomCenter,
                      colors: [
                        Colors.transparent,
                        Colors.black.withOpacity(0.6),
                      ],
                    ),
                  ),
                  padding: const EdgeInsets.symmetric(vertical: 16, horizontal:
                  child: const ControlBar(),
                ),
              ),
            ],
          ),
        );
      },
    );
  }
}

/// Video display widget that caches the video renderer to prevent re-rendering
class _VideoDisplayWidget extends StatefulWidget {
  final RoomContext roomCtx;

  const _VideoDisplayWidget({required this.roomCtx});

  @override
  State<_VideoDisplayWidget> createState() => _VideoDisplayWidgetState();
}
```

```
class _VideoDisplayWidgetState extends State<_VideoDisplayWidget> {
  lk.VideoTrackRenderer? _cachedVideoRenderer;
  String? _lastVideoTrackId;

  @override
  void initState() {
    super.initState();
    // Listen for track published events
    widget.roomCtx.room.addListener(_onRoomChanged);
  }

  @override
  void dispose() {
    widget.roomCtx.room.removeListener(_onRoomChanged);
    super.dispose();
  }

  void _onRoomChanged() {
    // Force rebuild when room state changes (e.g., new tracks published)
    if (mounted) {
      setState(() {});
    }
  }

  @override
  Widget build(BuildContext context) {
    // Get remote participants
    final remoteParticipants = widget.roomCtx.room.remoteParticipants.values.toL

    if (remoteParticipants.isEmpty) {
      return const Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            CircularProgressIndicator(
              valueColor: AlwaysStoppedAnimation<Color>(Colors.blue),
            ),
            SizedBox(height: 16),
            Text(
              'Waiting for AI Avatar to join...',
              style: TextStyle(color: Colors.white70, fontSize: 16),
            ),
            SizedBox(height: 8),
            Text(
              'Make sure the backend agent is running',
              style: TextStyle(color: Colors.white54, fontSize: 12),
            ),
          ],
        ),
      );
    }

    // Find the first remote participant with video
    for (final participant in remoteParticipants) {
      _logger.fine('🔍 Checking participant: ${participant.identity}');
      _logger.fine('   Video tracks count: ${participant.videoTrackPublications.
      
      // Check all video tracks, not just subscribed ones
      final videoTracks = participant.videoTrackPublications
          .where((pub) => pub.track != null)
          .toList();
      _logger.fine('   Available video tracks: ${videoTracks.length}');
      for (final pub in videoTracks) {
        _logger.fine('     Track: ${pub.sid}, enabled: ${pub.enabled}, subscribe
      }

      if (videoTracks.isNotEmpty) {
        final videoTrack = videoTracks.first.track as lk.VideoTrack;

        // Only recreate renderer if track ID changed
        if (_lastVideoTrackId != videoTrack.sid) {
          _logger.info('🎬 Creating new video renderer for ${participant.identit
          _cachedVideoRenderer = lk.VideoTrackRenderer(
            videoTrack,
            fit: lk.VideoViewFit.cover,
          );
          _lastVideoTrackId = videoTrack.sid;
        }

        return Container(
          color: Colors.black,
          child: _cachedVideoRenderer!,
        );
      }
    }

    return const Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            Icons.videocam_off,
            color: Colors.white54,
            size: 48,
          ),
          SizedBox(height: 16),
          Text(
            'AI Avatar connected but no video yet',
            style: TextStyle(color: Colors.white70, fontSize: 16),
          ),
          SizedBox(height: 8),
          Text(
            'Video will appear when AI starts speaking',
            style: TextStyle(color: Colors.white54, fontSize: 12),
          ),
        ],
      ),
    );
  }
}

/// Audio handler widget that manages audio without affecting video rendering
class _AudioHandlerWidget extends StatefulWidget {
  final RoomContext roomCtx;

  const _AudioHandlerWidget({required this.roomCtx});

  @override
  State<_AudioHandlerWidget> createState() => _AudioHandlerWidgetState();
}
```

```
class _AudioHandlerWidgetState extends State<_AudioHandlerWidget> {
  @override
  Widget build(BuildContext context) {
    // Get remote participants
    final remoteParticipants = widget.roomCtx.room.remoteParticipants.values.toL

    for (final participant in remoteParticipants) {
      final audioTracks = participant.audioTrackPublications
          .where((pub) => pub.track != null && pub.subscribed)
          .toList();

      if (audioTracks.isNotEmpty) {
        _logger.fine('🔊 Audio track active for ${participant.identity}');
        // Audio is handled automatically by LiveKit Components
        // We just need to ensure the track is subscribed
        break;
      }
    }

    return const SizedBox.shrink();
  }
}

/// Loading overlay that shows only when no remote video is available
class _LoadingOverlay extends StatefulWidget {
  @override
  State<_LoadingOverlay> createState() => _LoadingOverlayState();
}

class _LoadingOverlayState extends State<_LoadingOverlay> {
  int _lastParticipantCount = 0;

  @override
  Widget build(BuildContext context) {
    final roomContext = RoomContext.of(context);

    if (roomContext != null) {
      final remoteParticipants = roomContext.room.remoteParticipants.values;

      // Only log when state changes
      if (remoteParticipants.length != _lastParticipantCount) {
        _logger.fine('🔍 Loading overlay: ${remoteParticipants.length} remote pa
        _lastParticipantCount = remoteParticipants.length;
      }

      final hasRemoteVideo = remoteParticipants.any((participant) {
        return participant.videoTrackPublications.isNotEmpty;
      });

      if (!hasRemoteVideo) {
        return Container(
          color: Colors.black.withOpacity(0.8),
          child: const Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                CircularProgressIndicator(
                  valueColor: AlwaysStoppedAnimation<Color>(Colors.blue),
                ),
                SizedBox(height: 16),
                Text(
                  'Waiting for AI Avatar video...',
                  style: TextStyle(color: Colors.white70, fontSize: 16),
                ),
              ],
            ),
          ),
        );
      }
    }

    return const SizedBox.shrink();
  }
}

/// Control bar widget for media controls
class ControlBar extends StatelessWidget {
  const ControlBar({super.key});

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        // Microphone toggle
        IconButton(
          onPressed: () {
            // Toggle microphone logic here
          },
          icon: const Icon(Icons.mic, color: Colors.white),
          style: IconButton.styleFrom(
            backgroundColor: Colors.black.withOpacity(0.5),
            shape: const CircleBorder(),
          ),
        ),

        // Camera toggle
        IconButton(
          onPressed: () {
            // Toggle camera logic here
          },
          icon: const Icon(Icons.videocam, color: Colors.white),
          style: IconButton.styleFrom(
            backgroundColor: Colors.black.withOpacity(0.5),
            shape: const CircleBorder(),
          ),
        ),

        // End call
        IconButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          icon: const Icon(Icons.call_end, color: Colors.red),
          style: IconButton.styleFrom(
            backgroundColor: Colors.black.withOpacity(0.5),
            shape: const CircleBorder(),
          ),
        ),
      ],
    );
  }
```

```
}
```

📱 LiveKit Configuration

3. config/livekit_config.dart

```
import 'dart:convert';
import 'dart:math';
import 'package:http/http.dart' as http;

class LiveKitConfig {
  // LiveKit server configuration
  static const String serverUrl = 'wss://your-project.livekit.cloud';
  static const String? tokenEndpoint = 'http://localhost:3000/token';

  // Generate random room and participant names
  static String get roomName {
    const chars = 'abcdefghijklmnopqrstuvwxyz0123456789';
    final random = Random();
    return 'room-${String.fromCharCodes(Iterable.generate(12, (_) => chars.codeUn
  }

  static String get participantName {
    const chars = 'abcdefghijklmnopqrstuvwxyz0123456789';
    final random = Random();
    return 'user-${String.fromCharCodes(Iterable.generate(8, (_) => chars.codeUn
  }

  // Get JWT token from token server
  static Future<String> getToken() async {
    if (tokenEndpoint == null) {
      throw Exception('Token endpoint not configured. Please set LIVEKIT_TOKEN_EN
    }

    try {
      final response = await http.post(
        Uri.parse(tokenEndpoint!),
        headers: {'Content-Type': 'application/json'},
        body: jsonEncode({
          'room': roomName,
          'participant': participantName,
        }),
      );

      if (response.statusCode == 200) {
        final data = jsonDecode(response.body);
        return data['token'] as String;
      } else {
        throw Exception('Token server returned ${response.statusCode}: ${response
      }
    } catch (e) {
      throw Exception('Failed to get token: $e');
    }
  }
```

🔧 Logging Configuration

4. Logging Setup

```
import 'package:logging/logging.dart';

// Create logger instance
final _logger = Logger('BitHumanFlutter');

void main() {
  // Initialize logger (show info level and above)
  Logger.root.level = Level.INFO;
  Logger.root.onRecord.listen((record) {
    print('${record.level.name}: ${record.time}: ${record.message}');
  });

  runApp(const BitHumanFlutterApp());
```

Project Structure

```
lib/
├── main.dart                  # Complete app implementation with LiveKit Componen
├── config/
|   └── livekit_config.dart   # LiveKit configuration and token management
└── .env                       # Environment variables
```

Key Components

- **BitHumanFlutterApp**: Main app widget with LiveKit theme and dark mode
- **ConnectionScreen**: Handles token generation and automatic room joining
- **VideoRoomScreen**: Full-screen AI Avatar display using LiveKit Components
- **_VideoDisplayWidget**: Cached video renderer to prevent re-rendering during speech
- **_AudioHandlerWidget**: Separate audio handling to avoid video flashing
- **_LoadingOverlay**: Loading indicator when no remote video is available
- **ControlBar**: Floating media controls (mic, camera, end call)
- **LiveKitConfig**: Configuration and token management with random room names
- **Logger**: Structured logging with INFO level and above

Architecture Benefits

- **LiveKit Components**: Official UI components for better stability
- **Cached Video Rendering**: Prevents flashing during audio changes
- **Separated Audio/Video**: Independent handling prevents re-rendering issues
- **Structured Logging**: Clean console output with appropriate log levels
- **Auto-connection**: Seamless user experience with automatic room joining LocalVideoTrack? get loca

```
```

MediaService

Handles camera and microphone permissions:

```
class MediaService extends ChangeNotifier {
  // Permission management
  Future<bool> requestCameraPermission();
  Future<bool> requestMicrophonePermission();

  // Media controls
  void toggleCamera();
  void toggleMicrophone();
  void switchCamera();

  // State properties
  bool get cameraPermissionGranted;
  bool get microphonePermissionGranted;
}
```

VideoCallScreen

Main video call interface with AI avatar integration:

```
class VideoCallScreen extends StatefulWidget {
  // Displays remote avatar video
  // Shows local camera preview
  // Provides media controls
  // Handles connection status
}
```

Configuration

LiveKit Configuration

```
class LiveKitConfig {
  static const String serverUrl = 'wss://your-project.livekit.cloud';
  static const String? tokenEndpoint = 'http://localhost:3000/token';
  static const String roomName = 'flutter-avatar-room';
  static const String participantName = 'Flutter User';

  // Media settings
  static const int videoWidth = 1280;
  static const int videoHeight = 720;
  static const int videoFps = 30;
  static const int videoBitrate = 1_000_000;
}
```

Platform-Specific Setup

iOS (ios/Runner/Info.plist):

```
<key>NSCameraUsageDescription</key>
<string>This app needs camera access for video calls with AI avatar</string>
```

```
<key>NSMicrophoneUsageDescription</key>
<string>This app needs microphone access for voice interaction with AI avatar</s
```

Android (android/app/src/main/AndroidManifest.xml):

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

🐍 Backend Implementation

Python Backend Code

The Python backend uses LiveKit agents with bitHuman integration:

1. requirements.txt

```
livekit-agents==0.6.0
livekit-plugins-openai==0.6.0
livekit-plugins-silero==0.6.0
bithuman==0.1.0
flask==3.0.0
python-dotenv==1.0.0
```

2. agent.py (Complete Agent Implementation)

```python
import asyncio
import logging
import os
from typing import AsyncGenerator

import livekit.agents
from livekit.agents import JobContext, WorkerOptions, cli
from livekit.agents.voice_assistant import VoiceAssistant
from livekit.plugins import openai, silero
from livekit import rtc
import bithuman

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("flutter-bithuman-agent")

class BitHumanAvatarAgent:
    def __init__(self, ctx: JobContext):
        self.ctx = ctx
        self.agent = None

        # Get environment variables
        self.api_secret = os.getenv("BITHUMAN_API_SECRET")
        self.avatar_id = os.getenv("BITHUMAN_AVATAR_ID", "A33NZN6384")
        if not self.api_secret:
            raise ValueError("BITHUMAN_API_SECRET environment variable is require

    async def start(self):
        """Start the AI avatar agent"""
        try:
            logger.info("Starting bitHuman avatar agent...")

            # Initialize bitHuman avatar session
            bithuman_avatar = bithuman.AvatarSession(
                api_secret=self.api_secret,
                avatar_id=self.avatar_id,
            )

            # Configure AI voice assistant
            self.agent = VoiceAssistant(
                vad=openai.VAD(),  # Voice Activity Detection
                stt=openai.STT(),  # Speech-to-Text
                llm=openai.LLM(),  # Language Model
                tts=bithuman.TTS(),  # Text-to-Speech (bitHuman)
            )

            # Start the agent
            await self.agent.start()

            # Connect avatar to the room
            await bithuman_avatar.start(self.agent, room=self.ctx.room)

            logger.info("Flutter integration agent is ready and running")

            # Keep the agent running
            await self.agent.run()

        except Exception as e:
            logger.error(f"Error starting agent: {e}")
            raise

async def entrypoint(ctx: JobContext):
    """Main entry point for the LiveKit agent"""
    try:
        # Connect to the room
        await ctx.connect()

        # Create and start the agent
        agent = BitHumanAvatarAgent(ctx)
        await agent.start()

    except Exception as e:
        logger.error(f"Agent failed: {e}")
        raise

if __name__ == "__main__":
    # Configure worker options
    cli.run_app(
        WorkerOptions(
            entrypoint_fnc=entrypoint,
            # Increase memory limit for avatar processing
            job_memory_warn_mb=2000,
            # Keep some processes ready for faster startup
            num_idle_processes=2,
            # Allow more time for initialization
            initialize_process_timeout=180,
        )
    )
```

3. token_server.py (Token Generation Server)

```python
from flask import Flask, request, jsonify
from livekit import api
from datetime import timedelta
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

app = Flask(__name__)

# LiveKit configuration
LIVEKIT_API_KEY = os.getenv("LIVEKIT_API_KEY")
LIVEKIT_API_SECRET = os.getenv("LIVEKIT_API_SECRET")
LIVEKIT_URL = os.getenv("LIVEKIT_URL")

if not all([LIVEKIT_API_KEY, LIVEKIT_API_SECRET, LIVEKIT_URL]):
    raise ValueError("Missing required LiveKit environment variables")

@app.route('/token', methods=['POST'])
def create_token():
    """Generate a JWT token for LiveKit room access"""
    try:
        data = request.get_json() or {}
        room = data.get('room', 'flutter-avatar-room')
        identity = data.get('participant', 'Flutter User')

        # Create access token
        at = api.AccessToken(
            LIVEKIT_API_KEY,
            LIVEKIT_API_SECRET,
            identity=identity
        )

        # Add video grant (permission to join room)
        at.add_grant(api.VideoGrant(room_join=True, room=room))

        # Set token expiration (1 hour)
        at.ttl = timedelta(hours=1)

        # Generate JWT token
        token = at.to_jwt()

        return jsonify({
            'token': token,
            'server_url': LIVEKIT_URL
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

```
@app.route('/health', methods=['GET'])
def health_check():
    """Health check endpoint"""
    return jsonify({'status': 'healthy'})

if __name__ == '__main__':
    print("🚀 Starting LiveKit Token Server...")
    print(f"📡 Server URL: {LIVEKIT_URL}")
    print("🔑 Token endpoint: http://localhost:3000/token")
    print("❤️  Health check: http://localhost:3000/health")

    app.run(host='0.0.0.0', port=3000, debug=True)
```

4. .env.example

```
# bitHuman API Configuration
BITHUMAN_API_SECRET=sk_bh_your_secret_here
BITHUMAN_AVATAR_ID=A33NZN6384

# OpenAI API Configuration
OPENAI_API_KEY=sk-proj_your_key_here

# LiveKit Configuration
LIVEKIT_API_KEY=APIyour_key
LIVEKIT_API_SECRET=your_secret
LIVEKIT_URL=wss://your-project.livekit.cloud
```

5. run_backend.sh (Startup Script)

```
#!/bin/bash

echo "🚀 Starting Flutter + bitHuman Backend..."

# Check if .env exists
if [ ! -f .env ]; then
    echo "❌ .env file not found. Please copy .env.example to .env and configure
    exit 1
fi

# Load environment variables
export $(cat .env | grep -v '^#' | xargs)

# Check required environment variables
if [ -z "$BITHUMAN_API_SECRET" ]; then
    echo "❌ BITHUMAN_API_SECRET is required"
    exit 1
fi

if [ -z "$LIVEKIT_API_KEY" ]; then
    echo "❌ LIVEKIT_API_KEY is required"
    exit 1
fi

echo "✅ Environment variables loaded"
```

```
# Start token server in background
echo "🔑 Starting token server..."
python token_server.py &
TOKEN_PID=$!

# Wait a moment for token server to start
sleep 2

# Start the agent
echo "🤖 Starting LiveKit agent..."
python agent.py dev

# Cleanup on exit
trap "kill $TOKEN_PID" EXIT
```

Agent Architecture

The Python backend uses LiveKit agents with bitHuman integration:

```
async def entrypoint(ctx: JobContext):
    # Connect to LiveKit room
    await ctx.connect()

    # Initialize bitHuman avatar
    bithuman_avatar = bithuman.AvatarSession(
        api_secret=api_secret,
        avatar_id=avatar_id,
    )

    # Configure AI session
    session = AgentSession(
        llm=openai.realtime.RealtimeModel(voice="coral"),
        vad=silero.VAD.load()
    )

    # Start avatar and AI
    await bithuman_avatar.start(session, room=ctx.room)
    await session.start(agent=Agent(instructions=...), room=ctx.room)
```

Key Features

- **Avatar Integration**: Uses bitHuman Cloud Essence for avatar rendering
- **AI Conversation**: OpenAI Realtime API for natural language processing
- **Voice Activity Detection**: Silero VAD for conversation flow
- **Error Handling**: Comprehensive error handling and logging
- **Configuration**: Environment-based configuration

Environment Variables

```
# bitHuman API
BITHUMAN_API_SECRET=sk_bh_your_secret_here
BITHUMAN_AVATAR_ID=A33NZN6384

# OpenAI API
```

```
OPENAI_API_KEY=sk-proj_your_key_here
OPENAI_VOICE=coral

# LiveKit
LIVEKIT_API_KEY=APIyour_key
LIVEKIT_API_SECRET=your_secret
LIVEKIT_URL=wss://your-project.livekit.cloud
```

🔧 Advanced Configuration

Custom Avatar Integration

Using Avatar ID

```
bithuman_avatar = bithuman.AvatarSession(
    api_secret=api_secret,
    avatar_id="YOUR_AVATAR_ID",
)
```

Using Custom Image

```
bithuman_avatar = bithuman.AvatarSession(
    api_secret=api_secret,
    avatar_image="path/to/your/image.jpg",
)
```

AI Personality Customization

```
agent_instructions = """
You are a helpful AI assistant integrated with a Flutter mobile app.
Respond naturally and concisely to user questions.
Keep responses brief and engaging for mobile users.
You can help with general questions, provide information,
and have friendly conversations.
"""
```

Voice Customization

```
llm=openai.realtime.RealtimeModel(
    voice="coral",  # Options: alloy, echo, fable, onyx, nova, shimmer, coral
    model="gpt-4o-mini-realtime-preview",
)
```

Flutter UI Customization

Theme Customization

```
class AppTheme {
  static const Color primaryColor = Color(0xFF2196F3);
  static const Color videoBackgroundColor = Color(0xFF1E1E1E);
  static const Color controlActiveColor = Color(0xFF4CAF50);
}
```

Video Quality Settings

```
class LiveKitConfig {
  static const int videoWidth = 1280;
  static const int videoHeight = 720;
  static const int videoFps = 30;
  static const int videoBitrate = 1_000_000;
}
```

🧪 Testing and Debugging

Backend Testing

1. **Diagnostic Tool**:

   ```
   python diagnose.py
   ```

2. **LiveKit Playground**:
   - Start agent: `python agent.py dev`
   - Visit: https://agents-playground.livekit.io
   - Use same LiveKit credentials

3. **Console Testing**:

   ```
   python agent.py console
   ```

Flutter Testing

1. **Unit Tests**:

   ```
   flutter test
   ```

2. **Integration Tests**:

   ```
   flutter test integration_test/
   ```

3. **Manual Testing**:
   - Test on different devices
   - Test camera/microphone permissions
   - Test network connectivity
   - Test avatar loading

Debug Configuration

Backend Debug

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Flutter Debug

```
import 'package:logger/logger.dart';

final logger = Logger('MyApp');
logger.d('Debug message');
```

🚀 Deployment

Backend Deployment

Docker Deployment

```
FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
CMD ["python", "agent.py", "start"]
```

Cloud Deployment

- AWS ECS/Fargate
- Google Cloud Run
- Azure Container Instances
- Heroku
- Railway

Flutter Deployment

Mobile Deployment

```
# iOS
flutter build ios --release

# Android
flutter build apk --release
flutter build appbundle --release
```

Web Deployment

```
flutter build web --release
# Deploy to Firebase, Vercel, Netlify, etc.
```

🔍 Troubleshooting

Common Issues

Backend Issues

1. **"Avatar session failed"**
   - Check bitHuman API secret
   - Verify avatar ID exists
   - Check account access
2. **"Module not found"**
   ```
   pip install -r requirements.txt
   ```
3. **Connection timeouts**
   - Check LiveKit credentials
   - Verify network connectivity
   - Check firewall settings

Flutter Issues

1. **"No camera found"**
   - Check device permissions
   - Verify camera not in use
   - Test on different device
2. **"Connection failed"**
   - Verify LiveKit server URL
   - Check token validity
   - Ensure backend is running
3. **"Avatar not showing"**
   - Check backend logs

- Verify bitHuman API key
- Test with LiveKit Playground

Performance Optimization

Backend Optimization

```
WorkerOptions(
    job_memory_warn_mb=2000,  # Increase memory limit
    num_idle_processes=2,     # More idle processes
    initialize_process_timeout=180,  # Longer timeout
)
```

Flutter Optimization

- Use `const` constructors where possible
- Implement proper disposal of resources
- Optimize video quality based on device capabilities
- Use efficient state management

📚 API Reference

LiveKit Flutter SDK

Room Management

```
// Connect to room
await room.connect(serverUrl, token);

// Disconnect from room
await room.disconnect();

// Get participants
List<RemoteParticipant> participants = room.remoteParticipants.values.toList();
```

Media Tracks

```
// Get video track
RemoteVideoTrack? videoTrack = participant.videoTrackPublications.values
    .firstWhere((pub) => pub.track != null)?.track as RemoteVideoTrack?;

// Get audio track
RemoteAudioTrack? audioTrack = participant.audioTrackPublications.values
    .firstWhere((pub) => pub.track != null)?.track as RemoteAudioTrack?;
```

Media Controls

```
// Toggle microphone
await localParticipant.setMicrophoneEnabled(!isEnabled);

// Toggle camera
await localParticipant.setCameraEnabled(!isEnabled);
```

bitHuman Python SDK

Avatar Session

```
# Create avatar session
avatar = bithuman.AvatarSession(
    api_secret=api_secret,
    avatar_id=avatar_id,
)

# Start avatar
await avatar.start(session, room=room)
```

Agent Session

```
# Create agent session
session = AgentSession(
    llm=openai.realtime.RealtimeModel(voice="coral"),
    vad=silero.VAD.load()
)

# Start agent
await session.start(agent=Agent(instructions=...), room=room)
```

🎯 Best Practices

Security

- Use secure token generation
- Validate all inputs
- Implement proper error handling
- Use HTTPS in production

Performance

- Optimize video quality for device capabilities
- Implement proper resource disposal
- Use efficient state management
- Monitor memory usage

User Experience

- Provide clear loading states
- Handle errors gracefully
- Implement proper permissions flow
- Test on various devices

Code Quality

- Follow Flutter/Dart conventions
- Implement proper error handling
- Write comprehensive tests
- Document complex logic

🆘 Support and Resources

Documentation

- Flutter Documentation
- LiveKit Flutter SDK
- bitHuman Documentation
- LiveKit Agents Documentation

Community

- Discord Community
- GitHub Issues
- LiveKit Community

Examples

- Flutter Example
- Cloud Essence Example
- Expression Examples

📋 Implementation Checklist

Backend Setup

- [ ] Python virtual environment created
- [ ] Dependencies installed (livekit-agents, bithuman, flask)
- [ ] Environment variables configured (.env file)
- [ ] Token server running on port 3000
- [ ] LiveKit agent running and connected
- [ ] bitHuman API secret configured
- [ ] OpenAI API key configured

Frontend Setup

- [ ] Flutter project created
- [ ] livekit_components dependency added
- [ ] main.dart implemented with LivekitRoom
- [ ] ParticipantLoop configured for video rendering
- [ ] Audio playback enabled (showAudioTracks: true)

- [ ] Custom layout builder for full-screen video
- [ ] Loading overlay implemented
- [ ] Error handling added
- [ ] Debug logging enabled

Testing & Validation

- [ ] Backend agent connects to LiveKit
- [ ] Token server generates valid JWT tokens
- [ ] Flutter app connects to room
- [ ] AI avatar video displays correctly
- [ ] Audio plays from AI avatar
- [ ] Loading states work properly
- [ ] Error handling works
- [ ] Debug logs show expected output

🎯 Success Criteria

Your implementation is successful when:

1. ✅ Flutter app launches without errors
2. ✅ Backend agent connects to LiveKit room
3. ✅ AI avatar video appears in full screen
4. ✅ Audio plays from AI avatar
5. ✅ Loading indicator shows when no video
6. ✅ Error handling works for connection issues
7. ✅ Debug logs show proper track rendering

Ready to build? Start with the Flutter Example and follow the setup instructions!

< Previous
Real-time Events
🔔 INTEGRATIONS