PROJECT TRINETRA

Hybrid AI-Human Visual Assistance Platform for the Visually Impaired

Tagline: Your World, Described. By AI and by a Friend.

TABLE OF CONTENTS

---

1. EXECUTIVE SUMMARY

Project Name: PROJECT TRINETRA

Trinetra (Sanskrit for "Three Eyes") represents the fusion of three perspectives: AI sight, human compassion, and technology accessibility.

PROJECT TRINETRA is a mobile and web-based platform designed to serve as an intelligent visual interpreter for the 63 million visually impaired individuals in India. The platform uniquely combines artificial intelligence with human connection to provide real-time scene interpretation, navigation assistance, and emergency support.

Unlike existing assistive tools that rely solely on AI or human help, PROJECT TRINETRA bridges the gap by offering a hybrid intelligence model that provides:

- Instant AI-powered visual descriptions using computer vision
- Human assistance from trained volunteers/family members via live video
- Native language support for accessibility across India's linguistic diversity
- Low-latency, high-reliability assistance

Target Users: Visually impaired and low-vision individuals (primary); caregivers and family members (secondary)

Primary Markets: India (Phase 1), Southeast Asia (Phase 2)

---

2. PROBLEM STATEMENT & MARKET ANALYSIS

2.1 The Problem

For the visually impaired community in India, everyday tasks present significant challenges:

- **Navigation:** Crossing streets, public transportation, indoor wayfinding
- **Reading:** Bus schedules, product labels, bills, documents
- **Shopping:** Finding products, identifying prices, currency identification
- **Identification:** Objects, faces, colors, text in surroundings
- **Accessibility Gap:** Existing tools are expensive, limited in capability, or require constant human availability

2.2 Current Solutions (Limitations)

Solution Type	Capabilities	Limitations
Standalone AI Apps	Image recognition, OCR	Lack contextual understanding, no real-time human judgment
Human Assistance	Nuanced understanding, empathy	Not always available, expensive, inconsistent
Assistive Devices	Limited functionality	High cost, often proprietary

2.3 Market Opportunity

- **63 million** visually impaired individuals in India
- **900 million** smartphone users in India
- Growing smartphone penetration in rural India
- Potential TAM (Total Addressable Market): $2.5-5 billion annually
- Government schemes like SUGAMYA BHARAT support digital inclusion

2.4 Regulatory Environment

- **Rights of Persons with Disabilities (RPwD) Act, 2016** requires digital accessibility
- **Digital India Initiative** supports accessible technology solutions
- **NASSCOM's Inclusive Policy** encourages assistive technology startups

---

3. SOLUTION OVERVIEW

3.1 Platform Architecture

PROJECT TRINETRA operates in three integrated modes:

Mode 1: Aria AI Guide (Autonomous)

The user activates the AI mode via wake word or button. The system:

1. Captures real-time video/images
2. Processes through GPT-4o Vision for scene analysis
3. Returns natural language descriptions in the user's native language
4. Provides contextual guidance for navigation

Use Cases: Reading menus, checking weather, identifying objects, general navigation

Mode 2: Aria Connect (Human Assistance)

The user requests live assistance through a one-tap interface. The system:

1. Connects to trained volunteers or family members
2. Streams video in real-time using WebRTC
3. Volunteer provides real-time guidance
4. Session is recorded (with consent) for improvement

Use Cases: Emergency navigation, complex situations, shopping assistance

Mode 3: Trusted Network

- Pre-defined trusted contacts (family, friends, trained volunteers)
- Priority response system
- Scheduled assistance windows
- Session history and feedback

3.2 Core Value Proposition

- **Instant AI:** No waiting for human response
- **Human Touch:** Access to compassionate human assistance when needed
- **Native Language:** Supports Hindi, English, Telugu, Tamil, Kannada, Malayalam, Bengali, Marathi
- **Always Available:** 24/7 AI mode, scheduled human assistance
- **Affordable:** Free AI tier, affordable premium for human assistance
- **Privacy-Focused:** Encrypted video streaming, user-controlled data
- **Fast:** Low-latency response (< 2 seconds for AI, < 5 seconds for human connection)

---

4. KEY FEATURES (CORE + ENHANCED)

4.1 Core Features

1. Real-Time Scene Description

- Continuous video feed analysis
- Contextual, conversational descriptions
- Audio output with adjustable speed/clarity
- Offline mode for basic descriptions

2. OCR (Optical Character Recognition)

- Text detection and reading
- Document scanning
- Currency note identification
- Sign and label reading

3. Object & Currency Identification

- Real-time object detection
- Color identification
- Currency denomination recognition (Indian notes)
- Food item identification

4. Intelligent Navigation Cues

- Distance estimation (far, near, immediate)
- Direction guidance (left, right, ahead)
- Obstacle detection and alerts
- Pedestrian crossing detection

5. One-Tap Video Assistance

- Emergency connection to volunteers
- Family member connection
- Queue management system
- Session history and feedback

6. Trusted Network Management

- Add family members/caregivers
- Volunteer verification and rating system
- Scheduled assistance windows
- SOS emergency escalation

7. Low-Latency Streaming

- WebRTC for real-time video
- Bandwidth optimization for low-connectivity areas
- Adaptive bitrate streaming

4.2 Enhanced Features (Beyond Original)

1. Offline Scene Description (Local Model)

- Lightweight ML model for basic descriptions
- Works without internet connection
- Handles common scenarios (indoor/outdoor, day/night, crowds)

2. Voice Commands & Wake Words

- Multiple wake words (Project Trinetra, Hey Trinetra, Namaste)
- Contextual command recognition
- Command chaining (e.g., "Read and then find the exit")

3. Contextual Awareness

- Location-based services (which store, which street)
- Time-based responses (rush hour, business hours)
- User activity history for better suggestions

4. Emergency Response System

- One-button SOS activation
- Automatic location sharing with emergency contacts
- Integration with local emergency services (pilot phase)
- Voice-based emergency description

5. Accessibility Features

- Adjustable audio output speeds
- Voice customization (male/female, accents)
- Haptic feedback for blind + deaf users
- Braille display integration (future)

6. Community & Learning

- Volunteer training modules
- Feedback system to improve AI
- Community forum for tips and tricks
- Peer support groups

7. Analytics & Insights Dashboard

- Usage patterns (for volunteer coordinators)
- Common user queries (for AI improvement)
- Volunteer performance metrics
- Platform reliability monitoring

8. Predictive Assistance

- "You're approaching a busy intersection" alerts
- Recommend Aria Connect when AI confidence is low
- Proactive check-ins for frequent users

---

5. TECHNICAL ARCHITECTURE

5.1 System Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────┐
│                 USER INTERFACE LAYER                            │
│  (React Native Mobile App / Flutter App / Web Interface)        │
└─────────────────────────────────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────────┐
│                 SERVICE LAYER                                   │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ API Gateway & Load Balancer (AWS ALB/NGINX)               │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
                         │
          ┌──────────────┼──────────────┬──────────────┐
          │              │              │              │
          ▼              ▼              ▼              ▼
┌──────────────┐ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐
│  AI          │ │  Video       │ │  Auth &      │ │  Trusted     │
│  Service     │ │  Streaming   │ │  User        │ │  Network     │
│  (Python)    │ │  Service     │ │  Mgmt        │ │  Service     │
└──────────────┘ └──────────────┘ └──────────────┘ └──────────────┘
          │              │              │              │
          └──────────────┼──────────────┴──────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────────┐
│              DATA & MESSAGE LAYER                               │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ PostgreSQL (User Data, Sessions)                          │  │
│  │ Redis (Cache, Real-time State)                            │  │
│  │ MongoDB (Logs, Analytics)                                 │  │
│  │ S3 (Media Storage, Session Recordings)                    │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────────┐
│          EXTERNAL SERVICES & INTEGRATIONS                       │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ OpenAI API (GPT-4o Vision, Whisper)                       │  │
│  │ Google Cloud TTS (WaveNet)                                │  │
│  │ Twilio/AWS Chime (Video Calls)                            │  │
│  │ Geolocation APIs (Google Maps)                            │  │
│  │ ML Services (Custom Models)                               │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
```

5.2 Data Flow Diagram

AI Guidance Flow: User Activates → Speech Recognition → Intent Detection → Scene Analysis → Response Generation → Audio Output

Human Assistance Flow: User Requests → Availability Check → Volunteer Assignment → Video Streaming → Session Management → Feedback Collection

---

6. DETAILED TECH STACK

6.1 Frontend

Mobile (Primary Platform)

- **Framework:** React Native or Flutter
    - *Recommendation:* Flutter (better performance for real-time video, single codebase)
- **State Management:** Redux (React Native) or Provider/Riverpod (Flutter)
- **Real-time Communication:** WebRTC SDK (example: agora-react-native-rtc)
- **Voice Recognition:** Native APIs + OpenAI Whisper SDK
- **Text-to-Speech:** Google Cloud TTS SDK
- **Permissions Handling:** react-native-permissions or permission_handler (Flutter)
- **Navigation:** React Navigation (RN) or GetX/Riverpod (Flutter)

Web Interface

- **Framework:** React.js or Vue.js
    - *Recommendation:* React with TypeScript
- **Real-time Communication:** Socket.io for signaling, WebRTC for video
- **UI Components:** Material-UI or Tailwind CSS
- **State Management:** Redux Toolkit or Zustand
- **Audio Processing:** Web Audio API, Tone.js
- **Accessibility:** WCAG 2.1 AA compliance

6.2 Backend

Primary Language & Framework

- **Language:** Python 3.10+
- **Framework:** FastAPI (modern, fast, supports async)
- **Deployment:** Docker containers on Kubernetes

Core Services

1. Authentication & User Management Service

- **Framework:** FastAPI + SQLAlchemy
- **Database:** PostgreSQL
- **Authentication:** JWT tokens, OAuth 2.0
- **Security:** bcrypt for password hashing, CORS protection

2. AI Processing Service

- **Framework:** FastAPI + Celery (for async tasks)
- **AI Models:**
    - GPT-4o Vision (OpenAI API)
    - OpenAI Whisper (speech recognition)

    - Custom trained models (TensorFlow/PyTorch)
- **Image Processing:** OpenCV, Pillow
- **Task Queue:** Celery with Redis broker

3. Video Streaming & WebRTC Service

- **Signaling:** Socket.io with FastAPI-socketio
- **WebRTC SFU/MCU:** Janus Gateway or Kurento
- **Alternative:** AWS Chime or Twilio Video SDK
- **Bandwidth Optimization:** VP9/H.264 codecs, adaptive bitrate

4. Text-to-Speech Service

- **Primary:** Google Cloud Natural Language API
- **Fallback:** Azure Text-to-Speech or ElevenLabs
- **Caching:** Redis caching of synthesized audio

5. Notification Service

- **Service:** Firebase Cloud Messaging (FCM) for Android
- **Alternative:** APNs for iOS
- **Queuing:** Background job processing

6. Logging & Analytics Service

- **Logging:** ELK Stack (Elasticsearch, Logstash, Kibana)
- **Alternative:** DataDog or New Relic
- **Analytics:** Custom events to MongoDB

6.3 Database Schema

PostgreSQL Tables

```
1. users
   - user_id (PK), phone_number, email, preferred_language,
     created_at, updated_at, is_verified

2. sessions
   - session_id (PK), user_id (FK), session_type (AI/Human),
     start_time, end_time, transcript, status

3. trusted_contacts
   - contact_id (PK), user_id (FK), contact_type (Family/Volunteer),
     contact_info, relationship, verified

4. volunteer_profiles
   - volunteer_id (PK), user_id (FK), training_status,
     rating, total_sessions, languages_spoken

5. ai_interactions
   - interaction_id (PK), user_id (FK), query, response,
     confidence_score, category (Navigation/Reading/Object)

6. feedback
   - feedback_id (PK), session_id (FK), rating, comments,
     improvement_area
```

Redis Keys

- `user:{user_id}:session_state` - Current session data
- `user:{user_id}:contact_queue` - Priority queue of available volunteers
- `ai_responses_cache:{hash}` - Cached AI responses

MongoDB Collections

- `activity_logs` - User activity tracking
- `ai_improvements` - ML model feedback data
- `analytics_events` - Custom analytics events

6.4 External APIs & Services

Service	Purpose	Alternative
OpenAI GPT-4o Vision	Scene understanding	Google Cloud Vision
OpenAI Whisper	Speech recognition	Google Speech-to-Text
Google Cloud TTS	Text-to-speech	Azure TTS, Elevenlabs
Google Maps API	Location services	Mapbox, OpenStreetMap
Twilio	Video/SMS	AWS Chime, Agora
Firebase	Push notifications	AWS SNS, OneSignal
AWS S3	Media storage	Google Cloud Storage, Azure Blob

7. STEP-BY-STEP BUILD GUIDE

Phase 1: Foundation Setup (Days 1-2)

1.1 Environment Setup

bash

```bash
# Clone repository
git clone https://github.com/your-org/project-trinetra.git
cd project-trinetra

# Create Python virtual environment
python3.10 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install Python dependencies
pip install fastapi uvicorn python-dotenv psycopg2-binary redis sqlalchemy
pip install openai google-cloud-texttospeech pydantic
pip install celery python-socketio

# Install Node dependencies (if using Node for signaling)
npm init -y
npm install socket.io express cors

# Initialize database
createdb project_trinetra
psql project_trinetra < schema.sql
```

1.2 Project Structure

```
project-trinetra/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py
│   │   ├── config.py
│   │   ├── models/
│   │   ├── schemas/
│   │   ├── services/
│   │   │   ├── ai_service.py
│   │   │   ├── video_service.py
│   │   │   ├── user_service.py
│   │   │   ├── notification_service.py
│   │   │   └── tts_service.py
│   │   ├── routers/
│   │   │   ├── auth.py
│   │   │   ├── ai_guide.py
│   │   │   ├── video_assist.py
│   │   │   └── network.py
│   │   ├── middleware/
│   │   ├── utils/
│   │   └── database.py
│   ├── requirements.txt
│   ├── Dockerfile
│   └── .env.example
├── frontend/
│   ├── mobile/
│   │   ├── lib/
│   │   │   ├── main.dart
│   │   │   ├── screens/
│   │   │   ├── services/
│   │   │   └── widgets/
│   │   └── pubspec.yaml (if Flutter)
│   └── web/
│       ├── src/
│       │   ├── App.jsx
│       │   ├── components/
│       │   ├── services/
│       │   └── pages/
│       └── package.json
├── ml_models/
│   ├── local_scene_model.py
│   ├── trained_models/
│   └── training_scripts/
│
```

```
├── docs/
│   ├── API.md
│   ├── ARCHITECTURE.md
│   └── DEPLOYMENT.md
└── docker-compose.yml
```

1.3 Environment Variables

bash

```bash
# .env file
DATABASE_URL=postgresql://user:password@localhost/project_trinetra
REDIS_URL=redis://localhost:6379/0
OPENAI_API_KEY=your_key
GOOGLE_CLOUD_API_KEY=your_key
JWT_SECRET_KEY=your_secret_key
ENVIRONMENT=development
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:8000
```

Phase 2: Backend Development (Days 2-4)

2.1 Database Setup

python

```python
# backend/app/database.py
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import os

DATABASE_URL = os.getenv("DATABASE_URL")
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

2.2 User Model & Authentication

python

```python
# backend/app/models.py
from sqlalchemy import Column, Integer, String, DateTime, Boolean
from datetime import datetime
from app.database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    phone_number = Column(String, unique=True, index=True)
    email = Column(String, unique=True, index=True, nullable=True)
    password_hash = Column(String)
    preferred_language = Column(String, default="en")
    created_at = Column(DateTime, default=datetime.utcnow)
    is_verified = Column(Boolean, default=False)

# Similar models for Session, TrustedContact, Volunteer, etc.
```

2.3 API Routes - Authentication

python

```python
# backend/app/routers/auth.py
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from app.database import get_db
from app.models import User
from app.schemas import UserCreate, UserLogin
from app.utils.security import hash_password, verify_password, create_access_token

router = APIRouter(prefix="/api/auth", tags=["auth"])

@router.post("/register")
async def register(user: UserCreate, db: Session = Depends(get_db)):
    existing_user = db.query(User).filter(User.phone_number == user.phone_number).first()
    if existing_user:
        raise HTTPException(status_code=400, detail="User already exists")

    hashed_password = hash_password(user.password)
    db_user = User(
        phone_number=user.phone_number,
        password_hash=hashed_password,
        preferred_language=user.preferred_language
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    return {"message": "User registered successfully", "user_id": db_user.id}

@router.post("/login")
async def login(user: UserLogin, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.phone_number == user.phone_number).first()

    if not db_user or not verify_password(user.password, db_user.password_hash):
        raise HTTPException(status_code=401, detail="Invalid credentials")

    access_token = create_access_token(data={"sub": str(db_user.id)})
    return {"access_token": access_token, "token_type": "bearer"}
```

2.4 AI Service Implementation

python

```python
# backend/app/services/ai_service.py
import openai
import os
from google.cloud import texttospeech
import json

class AIService:
    def __init__(self):
        self.openai_api_key = os.getenv("OPENAI_API_KEY")
        self.gcp_tts_client = texttospeech.TextToSpeechClient()

    async def analyze_scene(self, image_base64: str, user_language: str = "en"):
        """
        Analyze scene using GPT-4o Vision
        """
        try:
            response = openai.ChatCompletion.create(
                model="gpt-4-vision-preview",
                messages=[
                    {
                        "role": "user",
                        "content": [
                            {
                                "type": "image_url",
                                "image_url": {
                                    "url": f"data:image/jpeg;base64,{image_base64}"
                                }
                            },
                            {
                                "type": "text",
                                "text": f"""Provide a detailed, practical description for a visually impaired person.
                                Language: {user_language}.
                                Include: main objects, layout, movement guidance, and any potential hazards.
                                Keep response concise and actionable."""
                            }
                        ]
                    }
                ],
                temperature=0.7,
                max_tokens=300
            )

            description = response.choices[0].message.content
            return {
                "status": "success",
                "description": description,
                "confidence": 0.95
            }

        except Exception as e:
            return {"status": "error", "message": str(e)}

    async def speech_to_text(self, audio_file_path: str, language: str = "en-US"):
        """
        Convert speech to text using Whisper
        """
        try:
            with open(audio_file_path, "rb") as audio_file:
                transcript = openai.Audio.transcribe(
                    model="whisper-1",
                    file=audio_file,
                    language=language
                )

            return {
                "status": "success",
                "text": transcript["text"]
            }

        except Exception as e:
            return {"status": "error", "message": str(e)}

    async def text_to_speech(self, text: str, language_code: str = "en-US"):
        """
        Convert text to speech using Google Cloud TTS
        """
        try:
            synthesis_input = texttospeech.SynthesisInput(text=text)

            voice = texttospeech.VoiceSelectionParams(
                language_code=language_code,
                ssml_gender=texttospeech.SsmlVoiceGender.NEUTRAL
            )

            audio_config = texttospeech.AudioConfig(
                audio_encoding=texttospeech.AudioEncoding.MP3
            )
            response = self.gcp_tts_client.synthesize_speech(
                input=synthesis_input,
                voice=voice,
                audio_config=audio_config
            )

            return {
                "status": "success",
                "audio_content": response.audio_content
            }

        except Exception as e:
            return {"status": "error", "message": str(e)}

ai_service = AIService()
```

2.5 AI Guidance Routes

python

```python
# backend/app/routers/ai_guide.py
from fastapi import APIRouter, File, UploadFile, Depends, HTTPException
from sqlalchemy.orm import Session
from app.database import get_db
from app.services.ai_service import ai_service
from app.utils.security import get_current_user
import base64

router = APIRouter(prefix="/api/ai", tags=["AI Guide"])

@router.post("/analyze-scene")
async def analyze_scene(
    file: UploadFile = File(...),
    db: Session = Depends(get_db),
    current_user = Depends(get_current_user)
):
    """Analyze scene from uploaded image"""
    try:
        # Read image
        contents = await file.read()
        image_base64 = base64.b64encode(contents).decode()

        # Call AI service
        result = await ai_service.analyze_scene(
            image_base64,
            current_user.preferred_language
        )

        # Generate audio response
        if result["status"] == "success":
            audio_result = await ai_service.text_to_speech(
                result["description"],
                current_user.preferred_language
            )

            result["audio_url"] = audio_result.get("audio_content")

        return result

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@router.post("/voice-command")
async def voice_command(
    file: UploadFile = File(...),
    db: Session = Depends(get_db),
    current_user = Depends(get_current_user)
):
    """Process voice command"""
    try:
        # Convert speech to text
        audio_path = f"/tmp/{file.filename}"
        with open(audio_path, "wb") as f:
            f.write(await file.read())

        transcription = await ai_service.speech_to_text(
            audio_path,
            current_user.preferred_language
        )

        if transcription["status"] == "success":
            # Process intent and respond
            response_text = await process_intent(transcription["text"])
            audio_result = await ai_service.text_to_speech(
                response_text,
                current_user.preferred_language
            )

            return {
                "status": "success",
                "command": transcription["text"],
                "response": response_text,
                "audio_url": audio_result.get("audio_content")
            }

        return transcription

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

async def process_intent(command: str):
    """Simple intent processing (can be enhanced with NLP)"""
    # This is a basic example - in production, use NLP library like spaCy
    command_lower = command.lower()

    if "read" in command_lower:
        return "Please point your camera at the text you want me to read."
    elif "describe" in command_lower:
        return "Please point your camera at your surroundings."
    elif "find" in command_lower:
        return "Connecting you with a volunteer for help finding that item."
    else:
        return "Command not understood. Please try again."
```

2.6 Video Streaming Service (WebRTC Signaling)

python

```python
# backend/app/services/video_service.py
import socketio
import json
from typing import Dict

sio = socketio.AsyncServer(
    async_mode='asgi',
    cors_allowed_origins='*'
)

active_sessions: Dict = {}

@sio.event
async def connect(sid, environ):
    print(f"Client connected: {sid}")

@sio.event
async def disconnect(sid):
    print(f"Client disconnected: {sid}")
    if sid in active_sessions:
        del active_sessions[sid]

@sio.event
async def offer(sid, data):
    """Handle WebRTC offer from initiator"""
    caller_id = data.get("caller_id")
    callee_id = data.get("callee_id")
    offer = data.get("offer")

    active_sessions[sid] = {
        "caller_id": caller_id,
        "callee_id": callee_id,
        "offer": offer
    }

    # Route to callee
    await sio.emit("offer", {"caller_id": caller_id, "offer": offer},
                   to=callee_id if callee_id in active_sessions else None)

@sio.event
async def answer(sid, data):
    """Handle WebRTC answer from receiver"""
    caller_id = data.get("caller_id")
    answer = data.get("answer")
    await sio.emit("answer", {"answer": answer}, to=caller_id)

@sio.event
async def ice_candidate(sid, data):
    """Handle ICE candidates"""
    target_id = data.get("target_id")
    candidate = data.get("candidate")

    await sio.emit("ice_candidate", {"candidate": candidate}, to=target_id)
```

2.7 Main FastAPI Application

python

```python
# backend/app/main.py
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
import os
from socketio import ASGIApp
from app.routers import auth, ai_guide, video_assist, network
from app.services.video_service import sio
from app.database import engine, Base

# Create tables
Base.metadata.create_all(bind=engine)

app = FastAPI(
    title="PROJECT TRINETRA API",
    description="Hybrid AI-Human Visual Assistance Platform",
    version="1.0.0"
)

# CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=os.getenv("ALLOWED_ORIGINS", "localhost").split(","),
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
app.include_router(auth.router)
app.include_router(ai_guide.router)
# app.include_router(video_assist.router)
# app.include_router(network.router)

# WebSocket for video signaling
app_with_socketio = ASGIApp(sio, app)

@app.get("/health")
async def health_check():
    return {"status": "ok", "service": "PROJECT TRINETRA API"}

if __name__ == "__main__":
```

```
import uvicorn
uvicorn.run(app, host="0.0.0.0", port=8000)
```

Phase 3: Frontend Development (Days 3-5)

3.1 Flutter Mobile App Setup

bash

```bash
# Create Flutter project
flutter create --org com.trinetra project_trinetra_mobile
cd project_trinetra_mobile

# Add dependencies to pubspec.yaml
flutter pub add:
  - dio (HTTP client)
  - provider (State management)
  - speech_to_text (STT)
  - flutter_tts (TTS)
  - camera (Camera access)
  - video_player (Video playback)
  - permission_handler (Permissions)
  - socket_io_client (WebSocket)
  - agora_rtc_engine (Video calling)
  - http
  - shared_preferences
```

3.2 Flutter Main App Structure

dart

```dart
// lib/main.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'screens/login_screen.dart';
import 'screens/home_screen.dart';
import 'screens/ai_guide_screen.dart';
import 'services/auth_service.dart';
import 'services/api_service.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => AuthService()),
        ChangeNotifierProvider(create: (_) => ApiService()),
      ],
      child: MaterialApp(
        title: 'Project Trinetra',
        theme: ThemeData(
          primarySwatch: Colors.blue,
          useMaterial3: true,
          visualDensity: VisualDensity.adaptivePlatformDensity,
        ),
        home: Consumer<AuthService>(
          builder: (context, authService, _) {
            if (authService.isAuthenticated) {
              return const HomeScreen();
            }
            return const LoginScreen();
          },
        ),
        routes: {
          '/home': (context) => const HomeScreen(),
          '/ai-guide': (context) => const AiGuideScreen(),
        },
      ),
    );
    }
  }
```

3.3 API Service (Dio Client)

dart

```dart
// lib/services/api_service.dart
import 'package:dio/dio.dart';
import 'package:shared_preferences/shared_preferences.dart';

class ApiService extends ChangeNotifier {
  late Dio _dio;
  final String baseUrl = 'http://your-api-url.com/api';
  String? _accessToken;

  ApiService() {
    _dio = Dio(BaseOptions(
      baseUrl: baseUrl,
      connectTimeout: const Duration(seconds: 30),
      receiveTimeout: const Duration(seconds: 30),
    ));

    _dio.interceptors.add(
      InterceptorsWrapper(
        onRequest: (options, handler) async {
          if (_accessToken != null) {
            options.headers['Authorization'] = 'Bearer $_accessToken';
          }
          return handler.next(options);
        },
        onError: (error, handler) {
          if (error.response?.statusCode == 401) {
            // Handle token refresh
          }
          return handler.next(error);
        },
      ),
    );
  }

  Future<void> setAccessToken(String token) async {
    _accessToken = token;
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('access_token', token);
    notifyListeners();
  }

  Future<Map<String, dynamic>> analyzeScene(String imagePath) async {
    try {
      FormData formData = FormData.fromMap({
```

```
      'file': await MultipartFile.fromFile(imagePath),
    });

    final response = await _dio.post('/ai/analyze-scene', data: formData);
    return response.data;
  } catch (e) {
    throw Exception('Failed to analyze scene: $e');
  }
}

Future<Map<String, dynamic>> voiceCommand(String audioPath) async {
  try {
    FormData formData = FormData.fromMap({
      'file': await MultipartFile.fromFile(audioPath),
    });

    final response = await _dio.post('/ai/voice-command', data: formData);
    return response.data;
  } catch (e) {
    throw Exception('Failed to process voice command: $e');
  }
}

Future<bool> register(String phoneNumber, String password, String language) async {
  try {
    final response = await _dio.post(
      '/auth/register',
      data: {
        'phone_number': phoneNumber,
        'password': password,
        'preferred_language': language,
      },
    );
    return response.statusCode == 200;
  } catch (e) {
    throw Exception('Registration failed: $e');
  }
}

Future<bool> login(String phoneNumber, String password) async {
  try {
    final response = await _dio.post(
      '/auth/login',
      data: {
          'phone_number': phoneNumber,
          'password': password,
        },
      );

      if (response.statusCode == 200) {
        await setAccessToken(response.data['access_token']);
        return true;
      }
      return false;
    } catch (e) {
      throw Exception('Login failed: $e');
    }
  }
}
```

3.4 Authentication Service

dart

```dart
// lib/services/auth_service.dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'api_service.dart';

class AuthService extends ChangeNotifier {
  bool _isAuthenticated = false;
  String? _userId;
  String? _phoneNumber;

  bool get isAuthenticated => _isAuthenticated;
  String? get userId => _userId;
  String? get phoneNumber => _phoneNumber;

  AuthService() {
    _checkAuthStatus();
  }

  Future<void> _checkAuthStatus() async {
    final prefs = await SharedPreferences.getInstance();
    _isAuthenticated = prefs.containsKey('access_token');
    _userId = prefs.getString('user_id');
    _phoneNumber = prefs.getString('phone_number');
    notifyListeners();
  }

  Future<bool> register(String phoneNumber, String password, String language) async {
    try {
      final apiService = ApiService();
      await apiService.register(phoneNumber, password, language);
      return true;
    } catch (e) {
      print('Registration error: $e');
      return false;
    }
  }

  Future<bool> login(String phoneNumber, String password) async {
    try {
      final apiService = ApiService();
      bool success = await apiService.login(phoneNumber, password);

      if (success) {
        final prefs = await SharedPreferences.getInstance();
        await prefs.setString('phone_number', phoneNumber);
        _phoneNumber = phoneNumber;
        _isAuthenticated = true;
        notifyListeners();
      }

      return success;
    } catch (e) {
      print('Login error: $e');
      return false;
    }
  }

  Future<void> logout() async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.clear();
    _isAuthenticated = false;
    _userId = null;
    _phoneNumber = null;
    notifyListeners();
  }
}
```

3.5 AI Guide Screen (Main Feature)

dart

```dart
// lib/screens/ai_guide_screen.dart
import 'package:flutter/material.dart';
import 'package:camera/camera.dart';
import 'package:speech_to_text/speech_to_text.dart' as stt;
import 'package:flutter_tts/flutter_tts.dart';
import 'package:provider/provider.dart';
import '../services/api_service.dart';
import 'dart:io';

class AiGuideScreen extends StatefulWidget {
  const AiGuideScreen({Key? key}) : super(key: key);

  @override
  State<AiGuideScreen> createState() => _AiGuideScreenState();
}

class _AiGuideScreenState extends State<AiGuideScreen> {
  CameraController? _cameraController;
  late stt.SpeechToText _speechToText;
  late FlutterTts _flutterTts;
  bool _isListening = false;
  bool _isProcessing = false;
  String _responseText = '';
  List<CameraDescription>? cameras;

  @override
  void initState() {
    super.initState();
    _initializeServices();
  }

  Future<void> _initializeServices() async {
    // Initialize camera
    cameras = await availableCameras();
    if (cameras != null && cameras!.isNotEmpty) {
      _cameraController = CameraController(
        cameras!.first,
        ResolutionPreset.medium,
      );
      await _cameraController!.initialize();
    }

    // Initialize speech to text
    _speechToText = stt.SpeechToText();
  await _speechToText.initialize(
    onError: (error) => print('Error: $error'),
    onStatus: (status) => print('Status: $status'),
  );

  // Initialize TTS
  _flutterTts = FlutterTts();
  await _flutterTts.setLanguage('en-US');
  await _flutterTts.setSpeechRate(0.5);

  if (mounted) {
    setState(() {});
  }
}

Future<void> _captureAndAnalyze() async {
  if (_cameraController == null || !_cameraController!.value.isInitialized) {
    return;
  }

  setState(() {
    _isProcessing = true;
    _responseText = 'Analyzing...';
  });

  try {
    XFile image = await _cameraController!.takePicture();
    final apiService = Provider.of<ApiService>(context, listen: false);
    final response = await apiService.analyzeScene(image.path);

    if (response['status'] == 'success') {
      setState(() {
        _responseText = response['description'];
      });

      // Speak the response
      await _flutterTts.speak(response['description']);
    }
  } catch (e) {
    setState(() {
      _responseText = 'Error analyzing scene: $e';
    });
    await _flutterTts.speak('Error analyzing scene');
  } finally {
    setState(() {
      _isProcessing = false;
    });
  }
}

Future<void> _startListening() async {
  if (!_isListening && _speechToText.isAvailable) {
    setState(() {
      _isListening = true;
      _responseText = 'Listening...';
    });

    _speechToText.listen(
      onResult: (result) async {
        if (result.finalResult) {
          setState(() {
            _isListening = false;
          });

          // Process voice command
          _processVoiceCommand(result.recognizedWords);
        }
      },
    );
  }
}

Future<void> _processVoiceCommand(String command) async {
  setState(() {
    _isProcessing = true;
    _responseText = 'Processing command...';
  });

  try {
    // In real implementation, send audio to backend
    // For now, simulate response
    String response = _generateResponse(command);

    setState(() {
      _responseText = response;
    });

    await _flutterTts.speak(response);
  } catch (e) {
    setState(() {
      _responseText = 'Error: $e';
    });
  } finally {
    setState(() {
      _isProcessing = false;
    });
  }
}

String _generateResponse(String command) {
  if (command.contains('read')) {
    return 'Point your camera at the text you want me to read.';
  } else if (command.contains('describe')) {
    return 'Please point your camera at your surroundings.';
  } else if (command.contains('find')) {
    return 'I can help you find that. Let me analyze your surroundings.';
  } else {
    return 'I understand you want to $command. Let me help you.';
  }
}

@override
void dispose() {
  _cameraController?.dispose();
  _speechToText.stop();
  _flutterTts.stop();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('AI Guide'),
      centerTitle: true,
    ),
    body: _cameraController?.value.isInitialized ?? false
        ? Column(
            children: [
              Expanded(
                flex: 2,
                child: CameraPreview(_cameraController!),
          ),
          Expanded(
            flex: 1,
            child: Container(
              color: Colors.grey[200],
              padding: const EdgeInsets.all(16.0),
              child: SingleChildScrollView(
                child: Text(
                  _responseText.isEmpty
                      ? 'Response will appear here'
                      : _responseText,
                  style: const TextStyle(fontSize: 16),
                ),
              ),
            ),
          ),
        ],
      )
    : const Center(child: CircularProgressIndicator()),
  floatingActionButton: Column(
    mainAxisAlignment: MainAxisAlignment.end,
    children: [
      FloatingActionButton(
        onPressed: _isProcessing ? null : _captureAndAnalyze,
        tooltip: 'Capture and analyze',
        child: const Icon(Icons.camera),
      ),
      const SizedBox(height: 16),
      FloatingActionButton(
        onPressed: _isProcessing ? null : _startListening,
        tooltip: 'Voice command',
        backgroundColor: _isListening ? Colors.red : Colors.blue,
        child: const Icon(Icons.mic),
      ),
    ],
  ),
 );
}
}
```

3.6 Home Screen

dart

```dart
// lib/screens/home_screen.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../services/auth_service.dart';
import 'ai_guide_screen.dart';

class HomeScreen extends StatelessWidget {
  const HomeScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Project Trinetra'),
        centerTitle: true,
        actions: [
          IconButton(
            icon: const Icon(Icons.logout),
            onPressed: () {
              Provider.of<AuthService>(context, listen: false).logout();
            },
          ),
        ],
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              Card(
                elevation: 4,
                child: Padding(
                  padding: const EdgeInsets.all(16.0),
                  child: Column(
                    children: [
                      const Icon(Icons.info, size: 48, color: Colors.blue),
                      const SizedBox(height: 16),
                      const Text(
                        'Welcome to Project Trinetra',
                        style: TextStyle(
                          fontSize: 20,
                          fontWeight: FontWeight.bold,
                        ),
        ),
        const SizedBox(height: 8),
        Text(
          'Your World, Described. By AI and by a Friend.',
          style: TextStyle(
            fontSize: 14,
            color: Colors.grey[600],
          ),
        ),
      ],
    ),
  ),
),
const SizedBox(height: 24),
_buildFeatureButton(
  context,
  'AI Guide',
  'Real-time scene description',
  Icons.camera,
  () => Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => const AiGuideScreen(),
    ),
  ),
),
const SizedBox(height: 16),
_buildFeatureButton(
  context,
  'Video Assistance',
  'Connect with a volunteer',
  Icons.videocam,
  () => ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Coming soon!')),
  ),
),
const SizedBox(height: 16),
_buildFeatureButton(
  context,
  'Trusted Network',
  'Manage your contacts',
  Icons.contacts,
  () => ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Coming soon!')),
```

```dart
              ),
            ),
          ],
        ),
      ),
    ),
  );
}

Widget _buildFeatureButton(
  BuildContext context,
  String title,
  String subtitle,
  IconData icon,
  VoidCallback onPressed,
) {
  return ElevatedButton(
    onPressed: onPressed,
    style: ElevatedButton.styleFrom(
      padding: const EdgeInsets.all(16),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(12),
      ),
    ),
    child: Column(
      children: [
        Icon(icon, size: 32),
        const SizedBox(height: 8),
        Text(
```