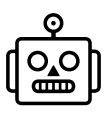




Let's Build Talking Lexi!



Complete 20-Day Project Plan

Team Roles & Responsibilities

Person 1: 3D Character & Scene (Frontend Lead)

- Three.js setup and 3D robot model
- Webcam overlay integration
- Lighting and visual effects
- Animation system architecture

Person 2: Voice & AI Integration (AI Lead)

- Web Speech API / Deepgram integration
- Gemini AI conversation logic
- Text-to-Speech [TTS] setup
- Voice command parsing

Person 3: Animation & Gestures (Animation Lead)

- Create animation library (wave, dance, jump, nod, backflip)
- Lip sync implementation
- MediaPipe Hands integration
- Gesture-to-animation mapping

Person 4: UI/UX & Integration (Full-stack Lead)

- User interface controls
- State management
- System integration
- Testing and debugging



Day-by-Day Breakdown

WEEK 1: Foundation (Days 1-7)

Day 1-2: Project Setup & Basic 3D

Person 1:

- Set up project structure (Vite + Three.js)
- Create basic Three.js scene with camera and lighting
- Add webcam background using video texture
- Test rendering on different devices

Person 2:

- Set up Web Speech API for voice recognition
- Test basic speech-to-text in browser
- Research Gemini API setup and get API key
- Create simple command detection (hello, hi, dance)

Person 3:

- Research 3D model sources (Sketchfab, Ready Player Me)
- Download/create simple robot model (GLTF format)
- Learn Three.js animation basics
- Set up animation mixer

Person 4:

- Create basic HTML structure
- Add start/stop buttons for voice control
- Set up state management (simple object or React state)
- Create project documentation structure

Milestone: Webcam shows in browser + basic 3D scene renders + voice recognition captures words

Day 3-4: Robot Model & Basic Animations

Person 1:

- Load robot GLTF model into scene
- Position and scale robot appropriately
- Add better lighting (ambient + directional)
- Create simple idle animation (breathing/hovering)

Person 2:

- Implement command pattern matching
- Set up Gemini API connection
- Create simple conversation flow
- Test AI responses

Person 3:

- Create 3 basic animations:
 - * Wave (arm movement)
 - * Nod (head movement)
 - * Jump (vertical translation)
- Test animation playback
- Set up animation queue system

Person 4:

- Create debug UI showing:
 - * Voice input text
 - * Detected commands
 - * Current animation state
- Add volume meter for microphone
- Implement basic error handling

Milestone: Robot appears on webcam background + can trigger 3 animations manually + voice commands detected

Day 5-6: Voice Command Integration

Person 1:

- Improve robot positioning/scaling
- Add glow effects or particles around robot
- Optimize scene performance
- Test on mobile browsers

Person 2:

- Connect voice commands to animation triggers
- Implement TTS (Web Speech API or alternative)
- Create conversation context (remember last 3 exchanges)
- Add wake word detection ("Hey Lexi")

Person 3:

- Add 2 more animations:
 - * Dance (body movement sequence)
 - * Backflip (rotation animation)
- Smooth animation transitions
- Create animation priority system

Person 4:

- Integrate all systems together
- Create main control flow
- Add visual feedback for voice activity
- Implement basic error recovery

Milestone: Say "wave" → Lexi waves + Say "hi" → Lexi responds with voice + animations trigger from commands

Day 7: Week 1 Integration & Testing

Everyone:

- Full system integration test
- Fix critical bugs
- Optimize performance
- Prepare demo for team review
- Document what works and what doesn't

Must work by end of Day 7:

- ☒ Robot visible on webcam background
- ☒ Voice commands trigger animations
- ☒ AI responds to questions
- ☒ TTS speaks responses
- ☒ At least 4 animations working

WEEK 2: Enhancement (Days 8-14)

Day 8-9: Lip Sync & Advanced Animations

Person 1:

- Improve visual quality (shadows, post-processing)
- Add background blur to webcam (focus on robot)
- Create "thinking" animation for when AI processes
- Add particle effects for special actions

Person 2:

- Implement basic lip sync:
 - * Detect audio volume
 - * Open/close mouth mesh
 - * Sync with TTS timing
- Improve conversation quality
- Add personality to responses

Person 3:

- Add MediaPipe Hands tracking
- Create hand gesture recognition:
 - * Thumbs up → compliment response
 - * Wave → wave back
 - * Peace sign → specific action
- Smooth gesture detection (avoid false positives)

Person 4:

- Create settings panel:
 - * Volume control
 - * Voice speed
 - * Animation speed
 - * Camera selection
- Improve UI aesthetics
- Add loading states

Milestone: Mouth moves when speaking + hands detected on webcam + gestures trigger responses

Day 10-11: Polish & Features

Person 1:

- Add environment (simple floor/background elements)
- Improve robot materials (metallic, glossy)
- Add camera movement (subtle follow robot)
- Performance optimization

Person 2:

- Expand command vocabulary (20+ commands)
- Add contextual responses
- Implement emotion detection in speech
- Add varied responses (not same answer twice)

Person 3:

- Create compound animations (wave + nod together)
- Add randomized idle variations
- Polish existing animations
- Add "signature move" (complex sequence)

Person 4:

- Add recording feature (capture session video)
- Create help/tutorial overlay
- Implement keyboard shortcuts
- Add analytics (track which commands used most)

Milestone: System feels polished + multiple ways to interact + personality emerges

Day 12-13: Advanced Features

Person 1:

- Add dynamic lighting based on time/mood
- Create alternate robot skins/colors
- Add screen effects (glitch, scan lines)
- Mobile optimization

Person 2:

- Add memory between sessions (localStorage - wait, we can't use localStorage!)
- Use in-memory state to remember conversation in session
- Create character personality traits
- Add joke/fun fact database

Person 3:

- Add physics-based movements (springy, realistic)
- Create reaction animations (surprised, happy, thinking)
- Add eye tracking (robot looks at hand gestures)
- Polish gesture recognition

Person 4:

- Create demo mode (auto-play features)
- Add accessibility features (text captions)
- Implement error recovery
- Cross-browser testing

Milestone: Feature-complete system ready for final polish

Day 14: Week 2 Integration & Testing

Everyone:

- Full system stress test
- Bug bash (find and fix 20+ bugs)
- Performance profiling
- User testing with friends/family
- Create feedback loop for improvements

WEEK 3: Polish & Demo Prep (Days 15-20)

Day 15-16: Polish & Optimization

Everyone focuses on:

- Visual polish (animations smooth, lighting beautiful)
- Audio quality (clear TTS, good recognition)
- Performance (60fps on target devices)
- UI refinement (intuitive, attractive)
- Bug fixing

Person 1: Visual excellence**Person 2:** Conversation quality**Person 3:** Animation perfection
Person 4: System stability

Day 17-18: Demo Preparation
Create demo script:

1. Introduction (30 sec)
 - "This is Lexi, an AI robot assistant"
 - Show webcam integration
2. Voice Interaction (60 sec)
 - "Hey Lexi, wave at the judges"
 - Ask a question, get AI response
 - Command dance
3. Gesture Control (45 sec)
 - Wave at camera → Lexi waves back
 - Thumbs up → Lexi responds positively
 - Show hand tracking visualization
4. Advanced Features (45 sec)
 - Lip sync demonstration
 - Complex animation (backflip)
 - Personality showcase (tell joke)
5. Q&A (variable)
 - Answer technical questions
 - Live interaction with judges

Everyone:

- Rehearse demo 10+ times
- Create backup plans:
 - * No internet → local mode
 - * Loud environment → typed commands
 - * Browser issues → video fallback
- Prepare presentation materials
- Test on expo hardware

Day 19: Final Testing

Morning:

- Test on 5+ different devices
- Test with different voices/accents
- Test in noisy environment
- Test with poor internet

Afternoon:

- Fix critical bugs only
- Create troubleshooting guide
- Prepare setup instructions

- Make backup builds

Evening:

- Freeze code (no more changes!)
- Final rehearsal
- Prepare physical demo booth
- Get good sleep!

Day 20: Expo Day

Setup (2 hours before):

- Test all equipment
- Run through demo twice
- Prepare for questions
- Relax and be confident!

During expo:

- Enthusiastic demos
- Engage with judges
- Handle errors gracefully
- Collect feedback

 **Technical Implementation Guide**

Core Technologies

```
// Project Structure
talking-lexi/
├── index.html
├── src/
│   ├── main.js           // Entry point
│   ├── scene/
│   │   ├── SceneManager.js // Three.js setup
│   │   ├── Robot.js       // Robot model & animations
│   │   └── WebcamBackground.js
│   ├── voice/
│   │   ├── SpeechRecognition.js
│   │   ├── TextToSpeech.js
│   │   └── CommandParser.js
│   ├── ai/
│   │   ├── GeminiClient.js // AI conversation
│   │   └── ResponseGenerator.js
│   ├── gestures/
│   │   ├── HandTracking.js // MediaPipe
│   │   └── GestureRecognizer.js
│   ├── ui/
│   │   ├── ControlPanel.js
│   │   └── StatusDisplay.js
│   └── assets/
│       ├── models/
│       │   ├── robot.glb
│       │   └── robot.fbx
│       ├── audio/
│       │   └── sounds/
```

Key Code Snippets

1. Three.js Scene Setup

```
audio/
└── sounds/
```

```
// SceneManager.js
import * as THREE from 'three';
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader';

class SceneManager {
  constructor(container) {
    this.container = container;
    this.scene = new THREE.Scene();
    this.camera = new THREE.PerspectiveCamera(
      75,
      window.innerWidth / window.innerHeight,
      0.1,
      1000
    );
    this.renderer = new THREE.WebGLRenderer({
      alpha: true,
      antialias: true
    });
    this.init();
  }

  init() {
    // Renderer setup
    this.renderer.setSize(window.innerWidth, window.innerHeight);
    this.renderer.setPixelRatio(window.devicePixelRatio);
    this.container.appendChild(this.renderer.domElement);

    // Camera position
    this.camera.position.z = 5;

    // Lighting
    const ambientLight = new THREE.AmbientLight(0xffffff, 0.6);
    this.scene.add(ambientLight);

    const directionalLight = new THREE.DirectionalLight(0xffffff, 0.8);
    directionalLight.position.set(10, 10, 5);
    this.scene.add(directionalLight);

    // Animation loop
    this.animate();
  }

  animate() {
    requestAnimationFrame(() => this.animate());
    this.renderer.render(this.scene, this.camera);
  }

  loadRobot(path) {
    const loader = new GLTFLoader();
    return new Promise((resolve, reject) => {
      loader.load(
        path,
        (gltf) => {
          this.robot = gltf.scene;
          this.scene.add(this.robot);
          resolve(this.robot);
        },
        undefined,
        reject
      );
    });
  }
}
```

2. Webcam Background

```
// WebcamBackground.js
class WebcamBackground {
  constructor(scene) {
    this.scene = scene;
    this.video = document.createElement('video');
  }

  async init() {
    const stream = await navigator.mediaDevices.getUserMedia({
      video: { width: 1280, height: 720 }
    });

    this.video.srcObject = stream;
    this.video.play();

    // Create video texture
    const texture = new THREE.VideoTexture(this.video);
    const geometry = new THREE.PlaneGeometry(16, 9);
    const material = new THREE.MeshBasicMaterial({
      map: texture,
      side: THREE.DoubleSide
    });

    const plane = new THREE.Mesh(geometry, material);
    plane.position.z = -10;
    this.scene.add(plane);
  }
}
```

3. Voice Recognition

```
// SpeechRecognition.js
class SpeechRecognition {
  constructor(onResult, onError) {
    this.recognition = new (window.SpeechRecognition ||
      window.webkitSpeechRecognition)();

    this.recognition.continuous = true;
    this.recognition.interimResults = true;

    this.recognition.onresult = (event) => {
      const transcript = Array.from(event.results)
        .map(result => result[0].transcript)
        .join('');
      onResult(transcript);
    };

    this.recognition.onerror = onError;
  }

  start() {
    this.recognition.start();
  }

  stop() {
    this.recognition.stop();
  }
}
```

4. Command Parser

```
// CommandParser.js
class CommandParser {
  constructor() {
    this.commands = {
      greetings: ['hi', 'hello', 'hey'],
      actions: {
        wave: ['wave', 'wave hand'],
        dance: ['dance', 'boogie'],
        jump: ['jump', 'leap'],
        nod: ['nod', 'yes'],
        backflip: ['backflip', 'flip']
      }
    };
  }

  parse(text) {
    text = text.toLowerCase();

    // Check for greetings
    if (this.commands.greetings.some(g => text.includes(g))) {
      return { type: 'greeting', action: 'wave' };
    }

    // Check for actions
    for (const [action, keywords] of Object.entries(this.commands.actions)) {
      if (keywords.some(k => text.includes(k))) {
        return { type: 'action', action };
      }
    }

    // Default to conversation
    return { type: 'conversation', text };
  }
}
```

5. Gemini AI Integration

```

// GeminiClient.js
class GeminiClient {
  constructor(apiKey) {
    this.apiKey = apiKey;
    this.endpoint = 'https://api.anthropic.com/v1/messages';
    this.conversationHistory = [];
  }

  async getResponse(userMessage) {
    this.conversationHistory.push({
      role: 'user',
      content: userMessage
    });

    try {
      const response = await fetch(this.endpoint, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          model: 'claude-sonnet-4-20250514',
          max_tokens: 1000,
          messages: this.conversationHistory,
          system: "You are Lexi, a friendly robot assistant. Keep responses
brief (1-2 sentences), enthusiastic, and fun. Use robot-themed language
occasionally."
        })
      });

      const data = await response.json();
      const assistantMessage = data.content[0].text;

      this.conversationHistory.push({
        role: 'assistant',
        content: assistantMessage
      });

      return assistantMessage;
    } catch (error) {
      console.error('AI Error:', error);
      return "Beep boop! My circuits are a bit scrambled. Try again!";
    }
  }
}

```

6. Text-to-Speech

```
// TextToSpeech.js
class TextToSpeech {
  constructor() {
    this.synth = window.speechSynthesis;
    this.voice = null;
    this.loadVoices();
  }

  loadVoices() {
    const voices = this.synth.getVoices();
    // Pick a robotic-sounding voice if available
    this.voice = voices.find(v => v.name.includes('Google')) || voices[0];
  }

  speak(text, onStart, onEnd) {
    const utterance = new SpeechSynthesisUtterance(text);
    utterance.voice = this.voice;
    utterance.rate = 1.1; // Slightly faster
    utterance.pitch = 1.2; // Slightly higher (robotic)

    utterance.onstart = onStart;
    utterance.onend = onEnd;

    this.synth.speak(utterance);
  }

  stop() {
    this.synth.cancel();
  }
}
```

7. Animation System

```

// Robot.js - Animation handling
class Robot {
  constructor(model) {
    this.model = model;
    this.mixer = new THREE.AnimationMixer(model);
    this.animations = {};
    this.currentAction = null;
  }

  loadAnimations(gltf) {
    gltf.animations.forEach(clip => {
      this.animations[clip.name] = this.mixer.clipAction(clip);
    });
  }

  playAnimation(name) {
    if (this.currentAction) {
      this.currentAction.fadeOut(0.5);
    }

    const action = this.animations[name];
    if (action) {
      action.reset().fadeIn(0.5).play();
      this.currentAction = action;
    }
  }

  update(delta) {
    this.mixer.update(delta);
  }

  // Simple lip sync
  syncLips(audioLevel) {
    if (this.model.morphTargetInfluences) {
      // Assuming morph target 0 is mouth open
      this.model.morphTargetInfluences[0] = audioLevel * 0.5;
    }
  }
}

```

8. MediaPipe Hands Integration

```
// HandTracking.js
import { Hands } from '@mediapipe/hands';
import { Camera } from '@mediapipe/camera_utils';

class HandTracking {
  constructor(videoElement, onResults) {
    this.hands = new Hands({
      locateFile: (file) => {
        return `https://cdn.jsdelivr.net/npm/@mediapipe/hands/${file}`;
      }
    });

    this.hands.setOptions({
      maxNumHands: 2,
      modelComplexity: 1,
      minDetectionConfidence: 0.5,
      minTrackingConfidence: 0.5
    });

    this.hands.onResults(onResults);

    this.camera = new Camera(videoElement, {
      onFrame: async () => {
        await this.hands.send({ image: videoElement });
      },
      width: 1280,
      height: 720
    });
  }

  start() {
    this.camera.start();
  }

  stop() {
    this.camera.stop();
  }
}
```

9. Gesture Recognition

```
// GestureRecognizer.js
class GestureRecognizer {
  recognizeGesture(landmarks) {
    if (!landmarks || landmarks.length === 0) return null;

    const hand = landmarks[0];

    // Thumbs up detection
    if (this.isThumbsUp(hand)) return 'thumbs_up';

    // Wave detection (hand moving side to side)
    if (this.isWaving(hand)) return 'wave';

    // Peace sign
    if (this.isPeaceSign(hand)) return 'peace';

    return null;
  }

  isThumbsUp(hand) {
    const thumb = hand[4];
    const index = hand[8];
    return thumb.y < index.y; // Thumb higher than index finger
  }

  isWaving(hand) {
    // Store previous positions and detect movement
    // Simplified version
    return false; // Implement based on velocity
  }

  isPeaceSign(hand) {
    const index = hand[8];
    const middle = hand[12];
    const ring = hand[16];
    const pinky = hand[20];

    // Index and middle extended, ring and pinky folded
    return (index.y < ring.y) && (middle.y < ring.y);
  }
}

```

10. Main Integration

```
// main.js
import { SceneManager } from './scene/SceneManager';
import { Robot } from './scene/Robot';
import { WebcamBackground } from './scene/WebcamBackground';
import { SpeechRecognition } from './voice/SpeechRecognition';
import { TextToSpeech } from './voice/TextToSpeech';
import { CommandParser } from './voice/CommandParser';
import { GeminiClient } from './ai/GeminiClient';
import { HandTracking } from './gestures/HandTracking';
import { GestureRecognizer } from './gestures/GestureRecognizer';

class TalkingLexi {
  constructor() {
    this.sceneManager = new SceneManager(document.getElementById('app'));
    this.tts = new TextToSpeech();
    this.commandParser = new CommandParser();
    this.ai = new GeminiClient('YOUR_API_KEY');
    this.gestureRecognizer = new GestureRecognizer();

    this.init();
  }

  async init() {
    // Load robot
    const robotModel = await this.sceneManager.loadRobot('/assets/models/robot.glb');
    this.robot = new Robot(robotModel);

    // Setup webcam
    const webcam = new WebcamBackground(this.sceneManager.scene);
    await webcam.init();

    // Setup voice recognition
    this.speech = new SpeechRecognition(
      (text) => this.handleVoiceInput(text),
      (error) => console.error('Speech error:', error)
    );

    // Setup hand tracking
    const video = webcam.video;
    this.handTracking = new HandTracking(video, (results) => {
      this.handleHandGestures(results);
    });

    // Start systems
    this.speech.start();
    this.handTracking.start();
  }

  async handleVoiceInput(text) {
    const command = this.commandParser.parse(text);

    if (command.type === 'action') {
      this.robot.playAnimation(command.action);
    } else if (command.type === 'greeting') {
      this.robot.playAnimation('wave');
      this.speak("Hey there! I'm Lexi!");
    } else {
      const response = await this.ai.getResponse(text);
      this.speak(response);
    }
  }

  handleHandGestures(results) {
    if (results.multiHandLandmarks) {
      const gesture = this.gestureRecognizer.recognizeGesture(
        results.multiHandLandmarks
      );

      if (gesture === 'thumbs_up') {
        this.robot.playAnimation('celebrate');
        this.speak("Thanks! You're awesome too!");
      } else if (gesture === 'wave') {
        this.robot.playAnimation('wave');
      } else if (gesture === 'peace') {
        this.robot.playAnimation('peace_sign');
      }
    }
  }

  speak(text) {
    this.tts.speak(
      text,
      () => {
        // Start lip sync
        this.isSpeaking = true;
      },
      () => {
        // Stop lip sync
        this.isSpeaking = false;
      }
    );
  }
}

// Initialize
const lexi = new TalkingLexi();
```

Where to Get Robot Models

Free 3D Models:

1. **Sketchfab** [sketchfab.com] - Filter "Downloadable" + "Free"
2. **Ready Player Me** - Generate custom avatars
3. **Mixamo** - Adobe's free character library
4. **TurboSquid Free** - Some free robot models
5. **CGTrader Free** - Community models

Create Your Own:

- **Blender** [free] - Model from scratch
- **MakeHuman** - Generate humanoid base
- **Vroid Studio** - Anime-style characters

Robot Model Requirements:

- Format: GLTF [.glb or .gltf]
- Rigged [has skeleton for animation]
- Under 10MB for performance
- Has face bones for lip sync [optional but nice]

Development Tools Needed

Required Software:

- **Code Editor:** VS Code
- **Browser:** Chrome [best WebGL support]
- **Version Control:** Git + GitHub
- **Package Manager:** npm or yarn

Useful Extensions:

- Live Server [VS Code]
- Three.js Snippets
- ESLint
- Prettier

Testing Tools:

- Chrome DevTools
- Mobile device [for testing]
- Different browsers [Firefox, Safari]

Success Metrics

By end of 20 days, you should have:

✓ **Core Features:**

- Robot renders on webcam background
- Voice commands trigger 5+ animations
- AI responds conversationally
- TTS speaks responses
- Hand gestures detected

✓ **Polish:**

- Smooth animations
- Clear audio
- Intuitive UI
- 60fps performance
- Mobile compatible

✓ **Demo Ready:**

- 3-minute demo script
- Backup plans for failures
- Wow moment [signature move]
- Judge interaction ready

Common Pitfalls & Solutions

Problem: Voice recognition doesn't work

Solution:

- Use Deepgram API as backup
- Add typed command input
- Test microphone permissions

Problem: 3D model won't load

Solution:

- Check file format (must be .glb)
- Verify file path
- Use simpler model temporarily
- Check console for errors

Problem: Animations are choppy

Solution:

- Reduce polygon count
- Use simpler textures
- Check frame rate
- Disable effects temporarily

Problem: Lip sync doesn't work

Solution:

- Use simpler mouth open/close
- Detect TTS audio level
- Fake it with timing

Problem: Hand tracking is slow

Solution:

- Lower model complexity in MediaPipe
- Process every other frame
- Use gestures as enhancement, not requirement

MVP vs Full Version

Minimum Viable Product (achievable by Day 10):

- Robot on webcam ✓
- 3 animations (wave, jump, nod) ✓
- Voice commands work ✓
- AI responds (text or speech) ✓

Full Version (target by Day 18):

- All animations polished ✓
- Hand gestures working ✓
- Lip sync ✓
- Beautiful visuals ✓
- Personality & humor ✓

Pro Tips

1. **Start simple, iterate fast** - Get basic working first, add features later
2. **Test early and often** - Don't wait until Day 19 to test on phones
3. **Use AI assistants heavily** - Claude/ChatGPT for code generation and debugging
4. **Commit code daily** - Use Git, don't lose work
5. **Record demo videos** - Document progress, use for backup if live fails
6. **Have fun with personality** - Make Lexi funny, quirky, memorable
7. **Prepare for Murphy's Law** - Everything that can go wrong, will. Have backups.

Final Demo Script

[0:00-0:30] Introduction
"Hi judges! This is Lexi, an AI-powered 3D robot assistant that combines voice recognition, computer vision, and AI conversation into one interactive experience."

[0:30-1:00] Voice Commands
"Hey Lexi, can you wave at the judges?"
→ Robot waves
"Lexi, do a backflip!"
→ Robot backflips

[1:00-1:30] AI Conversation
"Lexi, what's your favorite thing about being a robot?"
→ AI generates response, robot speaks with lip sync

[1:30-2:00] Hand Gestures
Wave at camera → Lexi waves back
Thumbs up → Lexi responds positively

[2:00-2:30] Showcase Features
"Notice the lip sync, the smooth animations, and how everything runs in a web browser"

[2:30-3:00] Q&A
Let judges interact with Lexi directly

☒ Your Action Items RIGHT NOW

Today:

- 1. ☒ Assign team roles
- 2. ☒ Set up GitHub repository
- 3. ☒ Create project structure
- 4. ☒ Schedule daily standup time (15 min)
- 5. ☒ Each person: Set up development environment