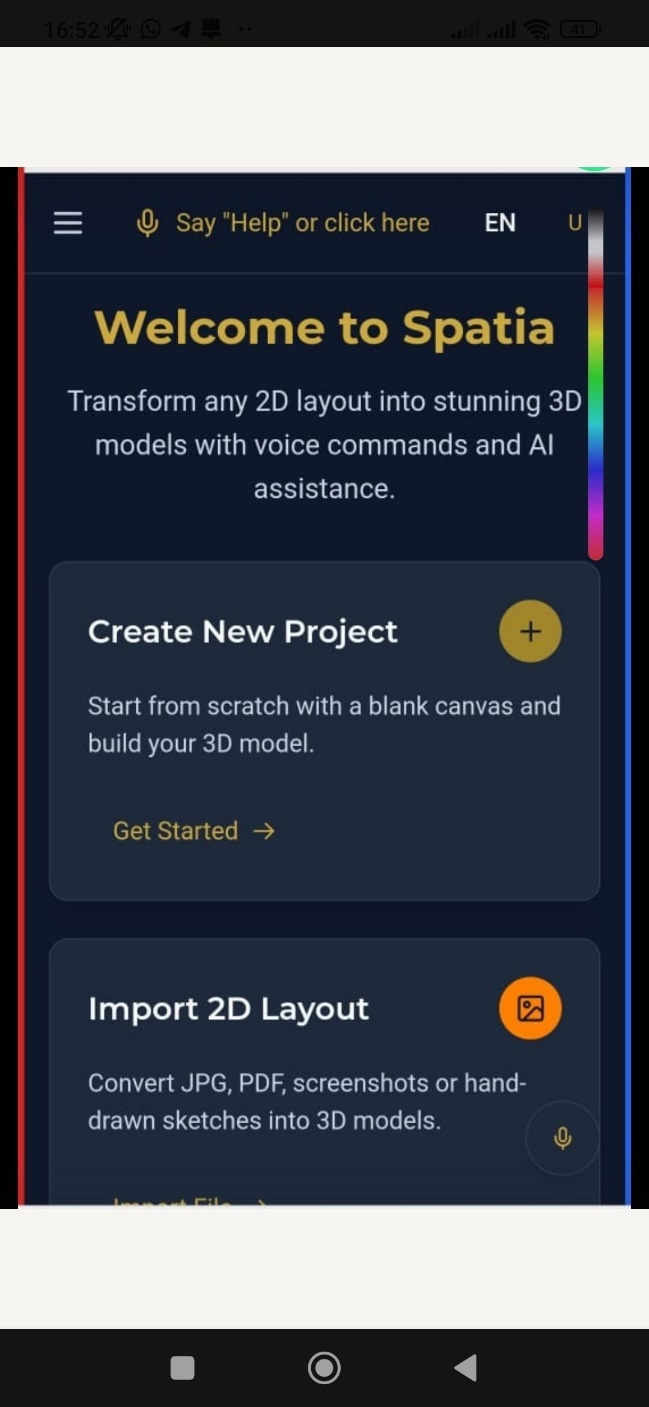
**Spatia - 2D to 3D Model Animation Conversion App Development Plan**

**LOGO **

****

**Thia Front End ,Vision and Layout what Spatia all about and what to achieve**

our requirements, You will outline a comprehensive plan to develop the Spatia mobile application that converts 2D layouts to 3D animations with AI-powered features.

**Core Features**

1. **2D to 3D Conversion**:
   * Accepts JPG, PNG, PDF, and sketch inputs
   * Generates 3D animated models from 2D layouts
2. **AI Integration**:
   * Voice command processing (Whisper AI)
   * Natural language understanding (GPT)
   * Automated architectural design capabilities
3. **Subscription Model**:
   * Free trial (1 conversion)
   * Monthly (15),6−month(15),6−*month*(70), and annual ($120) subscriptions
   * Bank payment integration (BCA Indonesia)

**Technical Implementation**

**Mobile App Architecture**

Copy

Download

┌─────────────────────────────────────────────────┐

│ Mobile App │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────┐ │

│ │ UI/UX │ │ Business │ │ State │ │

│ │ Components │ │ Logic │ │ Manager │ │

│ └─────────────┘ └─────────────┘ └─────────┘ │

└─────────────────────────────────────────────────┘

│ │ │

▼ ▼ ▼

┌─────────────────────────────────────────────────┐

│ Native Bridge │

└─────────────────────────────────────────────────┘

│ │ │

▼ ▼ ▼

┌─────────────────────────────────────────────────┐

│ Backend API Services │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────┐ │

│ │ Auth/User │ │ File │ │ Payment │ │

│ │ Management │ │ Processing │ │ Gateway │ │

│ └─────────────┘ └─────────────┘ └─────────┘ │

└─────────────────────────────────────────────────┘

│ │ │

▼ ▼ ▼

┌─────────────────────────────────────────────────┐

│ AI/ML Services │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────┐ │

│ │ 2D-to-3D │ │ Voice │ │ Design │ │

│ │ Conversion │ │ Processing │ │ AI │ │

│ └─────────────┘ └─────────────┘ └─────────┘ │

└─────────────────────────────────────────────────┘

**Technology Stack**

**Frontend (Mobile)**:

* React Native (cross-platform for iOS and Android)
* Three.js for 3D model rendering
* Expo for development and testing

**Backend**:

* Node.js with Express.js
* MongoDB for database
* AWS S3 for file storage

**AI Services**:

* Custom CNN + GAN model for 2D-to-3D conversion
* OpenAI Whisper for speech-to-text
* GPT for natural language processing

**Development Phases**

**Phase 1: Core Functionality (4-6 weeks)**

1. Set up React Native project with Expo
2. Implement basic UI with file upload functionality
3. Develop backend API for file processing
4. Integrate basic 2D-to-3D conversion using pre-trained models
5. Implement user authentication

**Phase 2: Advanced Features (4 weeks)**

1. Voice command integration
2. AI guidance system (text and voice)
3. Payment gateway integration
4. Subscription management
5. Enhanced 3D rendering capabilities

**Phase 3: Polish & Testing (2-3 weeks)**

1. Performance optimization
2. Cross-platform testing
3. User feedback implementation
4. App store submission preparation

**Detailed Implementation**

**1. React Native App Structure**

javascript

Copy

Download

*// App.js*

import React from 'react';

import { NavigationContainer } from '@react-navigation/native';

import { createStackNavigator } from '@react-navigation/stack';

import HomeScreen from './screens/HomeScreen';

import UploadScreen from './screens/UploadScreen';

import ModelViewer from './screens/ModelViewer';

import PaymentScreen from './screens/PaymentScreen';

const Stack = createStackNavigator();

export default function App() {

return (

<NavigationContainer>

<Stack.Navigator initialRouteName="Home">

<Stack.Screen name="Home" component={HomeScreen} />

<Stack.Screen name="Upload" component={UploadScreen} />

<Stack.Screen name="ModelViewer" component={ModelViewer} />

<Stack.Screen name="Payment" component={PaymentScreen} />

</Stack.Navigator>

</NavigationContainer>

);

}

**2. File Upload and Processing**

javascript

Copy

Download

*// UploadScreen.js*

import React, { useState } from 'react';

import { View, Text, Button, ActivityIndicator, Alert } from 'react-native';

import \* as ImagePicker from 'expo-image-picker';

import \* as FileSystem from 'expo-file-system';

import axios from 'axios';

export default function UploadScreen({ navigation }) {

const [isProcessing, setIsProcessing] = useState(false);

const pickImage = async () => {

*// Check subscription status first*

const hasSubscription = await checkSubscription();

if (!hasSubscription && await checkTrialUsed()) {

Alert.alert(

'Trial Used',

'You have used your free trial. Please upgrade to continue.',

[{ text: 'OK', onPress: () => navigation.navigate('Payment') }]

);

return;

}

let result = await ImagePicker.launchImageLibraryAsync({

mediaTypes: ImagePicker.MediaTypeOptions.Images,

allowsEditing: true,

quality: 1,

});

if (!result.cancelled) {

convertTo3D(result.uri);

}

};

const convertTo3D = async (imageUri) => {

setIsProcessing(true);

try {

const uploadResponse = await FileSystem.uploadAsync(

'https://your-api-endpoint/upload',

imageUri,

{

httpMethod: 'POST',

uploadType: FileSystem.FileSystemUploadType.MULTIPART,

fieldName: 'file',

headers: {

'Authorization': `Bearer ${await getAuthToken()}`,

},

}

);

const responseData = JSON.parse(uploadResponse.body);

navigation.navigate('ModelViewer', { modelUrl: responseData.modelUrl });

} catch (error) {

Alert.alert('Error', 'Failed to convert image to 3D model');

} finally {

setIsProcessing(false);

}

};

return (

<View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>

<Text>Upload your 2D layout</Text>

{isProcessing ? (

<ActivityIndicator size="large" />

) : (

<Button title="Select Image" onPress={pickImage} />

)}

</View>

);

}

**3. Payment Integration**

javascript

Copy

Download

*// PaymentScreen.js*

import React, { useState } from 'react';

import { View, Text, Button, StyleSheet } from 'react-native';

import axios from 'axios';

const plans = [

{ id: 'monthly', name: 'Monthly', price: '$15', days: 30 },

{ id: '6month', name: '6 Months', price: '$70', days: 180 },

{ id: 'yearly', name: 'Yearly', price: '$120', days: 365 },

];

export default function PaymentScreen({ navigation }) {

const [selectedPlan, setSelectedPlan] = useState(null);

const [isProcessing, setIsProcessing] = useState(false);

const handlePayment = async () => {

if (!selectedPlan) return;

setIsProcessing(true);

try {

const response = await axios.post('https://your-api-endpoint/payment', {

plan: selectedPlan,

userId: await getUserId(),

});

if (response.data.success) {

Alert.alert('Success', 'Payment processed successfully!');

navigation.goBack();

}

} catch (error) {

Alert.alert('Error', 'Payment processing failed');

} finally {

setIsProcessing(false);

}

};

return (

<View style={styles.container}>

<Text style={styles.title}>Choose Your Plan</Text>

{plans.map(plan => (

<View

key={plan.id}

style={[

styles.planCard,

selectedPlan === plan.id && styles.selectedPlan

]}

>

<Text style={styles.planName}>{plan.name}</Text>

<Text style={styles.planPrice}>{plan.price}</Text>

<Button

title="Select"

onPress={() => setSelectedPlan(plan.id)}

disabled={selectedPlan === plan.id}

/>

</View>

))}

<Button

title="Complete Payment"

onPress={handlePayment}

disabled={!selectedPlan || isProcessing}

/>

</View>

);

}

const styles = StyleSheet.create({

container: { padding: 20 },

title: { fontSize: 24, marginBottom: 20 },

planCard: {

padding: 15,

marginBottom: 15,

borderWidth: 1,

borderRadius: 5

},

selectedPlan: { borderColor: 'blue', backgroundColor: '#f0f8ff' },

planName: { fontSize: 18 },

planPrice: { fontSize: 16, fontWeight: 'bold' },

});

**4. 3D Model Viewer**

javascript

Copy

Download

*// ModelViewer.js*

import React, { useState, useEffect } from 'react';

import { View, Text, StyleSheet, Dimensions } from 'react-native';

import { WebView } from 'react-native-webview';

import ModelControls from '../components/ModelControls';

export default function ModelViewer({ route }) {

const { modelUrl } = route.params;

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

const htmlContent = `

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">

<style>

body { margin: 0; overflow: hidden; }

canvas { width: 100%; height: 100%; display: block; }

</style>

</head>

<body>

<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/three@0.128.0/examples/js/loaders/GLTFLoader.js"></script>

<script>

let scene, camera, renderer, model;

function init() {

// Scene setup

scene = new THREE.Scene();

scene.background = new THREE.Color(0xdddddd);

// Camera

camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);

camera.position.z = 5;

// Renderer

renderer = new THREE.WebGLRenderer({ antialias: true });

renderer.setSize(window.innerWidth, window.innerHeight);

document.body.appendChild(renderer.domElement);

// Lights

const light1 = new THREE.AmbientLight(0xffffff, 0.5);

scene.add(light1);

const light2 = new THREE.DirectionalLight(0xffffff, 0.5);

light2.position.set(0, 1, 1);

scene.add(light2);

// Load model

const loader = new THREE.GLTFLoader();

loader.load(

'${modelUrl}',

function (gltf) {

model = gltf.scene;

scene.add(model);

// Auto-rotate

function animate() {

requestAnimationFrame(animate);

if (model) model.rotation.y += 0.005;

renderer.render(scene, camera);

}

animate();

// Notify React Native that loading is complete

window.ReactNativeWebView.postMessage('loaded');

},

undefined,

function (error) {

window.ReactNativeWebView.postMessage('error:' + error.message);

}

);

}

function handleRotation(x, y) {

if (model) {

model.rotation.x = x;

model.rotation.y = y;

}

}

window.addEventListener('resize', function() {

camera.aspect = window.innerWidth / window.innerHeight;

camera.updateProjectionMatrix();

renderer.setSize(window.innerWidth, window.innerHeight);

});

window.addEventListener('message', function(event) {

const data = JSON.parse(event.data);

if (data.type === 'rotate') {

handleRotation(data.x, data.y);

}

});

init();

</script>

</body>

</html>

`;

return (

<View style={styles.container}>

{error ? (

<Text style={styles.error}>{error}</Text>

) : (

<>

<WebView

source={{ html: htmlContent }}

style={styles.webview}

onMessage={(event) => {

const msg = event.nativeEvent.data;

if (msg === 'loaded') {

setLoading(false);

} else if (msg.startsWith('error:')) {

setError(msg.substring(6));

}

}}

/>

{loading && (

<View style={styles.loadingContainer}>

<Text>Loading 3D model...</Text>

</View>

)}

<ModelControls />

</>

)}

</View>

);

}

const styles = StyleSheet.create({

container: { flex: 1 },

webview: { flex: 1 },

loadingContainer: {

...StyleSheet.absoluteFillObject,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: 'rgba(255,255,255,0.7)',

},

error: { padding: 20, color: 'red' },

});

**5. Voice Command Integration**

javascript

Copy

Download

*// VoiceCommand.js*

import React, { useState, useEffect } from 'react';

import { View, Text, TouchableOpacity, StyleSheet } from 'react-native';

import { Audio } from 'expo-av';

import \* as Speech from 'expo-speech';

export default function VoiceCommand({ onCommand }) {

const [isListening, setIsListening] = useState(false);

const [lastCommand, setLastCommand] = useState('');

const startListening = async () => {

try {

await Audio.requestPermissionsAsync();

setIsListening(true);

setLastCommand('');

*// Initialize speech recognition*

Speech.speak("I'm listening...");

const subscription = Speech.Recognition.addListener(({ results }) => {

if (results && results.length > 0) {

const command = results[0][0].transcript;

setLastCommand(command);

onCommand(command);

setIsListening(false);

subscription.remove();

}

});

await Speech.Recognition.start();

} catch (error) {

console.error('Speech recognition error', error);

setIsListening(false);

}

};

const stopListening = async () => {

await Speech.Recognition.stop();

setIsListening(false);

};

return (

<View style={styles.container}>

<TouchableOpacity

style={[styles.button, isListening && styles.listening]}

onPress={isListening ? stopListening : startListening}

>

<Text style={styles.buttonText}>

{isListening ? 'Listening...' : 'Voice Command'}

</Text>

</TouchableOpacity>

{lastCommand ? (

<Text style={styles.commandText}>Last command: {lastCommand}</Text>

) : null}

</View>

);

}

const styles = StyleSheet.create({

container: { alignItems: 'center', marginVertical: 10 },

button: {

backgroundColor: '#4285F4',

padding: 15,

borderRadius: 50,

width: 150,

alignItems: 'center',

},

listening: { backgroundColor: '#DB4437' },

buttonText: { color: 'white', fontWeight: 'bold' },

commandText: { marginTop: 10, color: '#666' },

});

**Backend API Implementation**

**1. User Authentication**

javascript

Copy

Download

*// authController.js*

const jwt = require('jsonwebtoken');

const User = require('../models/User');

const { JWT\_SECRET } = process.env;

exports.register = async (req, res) => {

try {

const { email, password } = req.body;

*// Check if user exists*

const existingUser = await User.findOne({ email });

if (existingUser) {

return res.status(400).json({ message: 'User already exists' });

}

*// Create new user*

const user = new User({ email, password });

await user.save();

*// Generate token*

const token = jwt.sign({ userId: user.\_id }, JWT\_SECRET, { expiresIn: '7d' });

res.status(201).json({ token, userId: user.\_id });

} catch (error) {

res.status(500).json({ message: 'Registration failed', error });

}

};

exports.login = async (req, res) => {

try {

const { email, password } = req.body;

*// Find user*

const user = await User.findOne({ email });

if (!user) {

return res.status(401).json({ message: 'Invalid credentials' });

}

*// Check password*

const isMatch = await user.comparePassword(password);

if (!isMatch) {

return res.status(401).json({ message: 'Invalid credentials' });

}

*// Generate token*

const token = jwt.sign({ userId: user.\_id }, JWT\_SECRET, { expiresIn: '7d' });

res.json({ token, userId: user.\_id });

} catch (error) {

res.status(500).json({ message: 'Login failed', error });

}

};

**2. File Processing and 3D Conversion**

javascript

Copy

Download

*// conversionController.js*

const fs = require('fs');

const path = require('path');

const { v4: uuidv4 } = require('uuid');

const axios = require('axios');

const FormData = require('form-data');

const Project = require('../models/Project');

const { uploadToS3, generateS3Url } = require('../utils/s3');

exports.processUpload = async (req, res) => {

try {

*// Check user's subscription status*

const user = await User.findById(req.userId).populate('subscription');

if (!user.canConvert()) {

return res.status(403).json({

message: 'Subscription required',

requiresPayment: true

});

}

*// Process file upload*

const file = req.file;

const fileId = uuidv4();

const fileExt = path.extname(file.originalname);

const fileName = `${fileId}${fileExt}`;

*// Upload to S3*

await uploadToS3(file.buffer, fileName);

*// Call AI service for conversion*

const formData = new FormData();

formData.append('file', file.buffer, { filename: fileName });

const aiResponse = await axios.post(process.env.AI\_SERVICE\_URL, formData, {

headers: formData.getHeaders(),

});

*// Save converted model to S3*

const modelUrl = await generateS3Url(aiResponse.data.model);

*// Create project record*

const project = new Project({

user: req.userId,

originalFile: fileName,

model3d: modelUrl,

});

await project.save();

*// Update user's conversion count*

await user.updateConversionCount();

res.json({

success: true,

modelUrl,

projectId: project.\_id,

conversionsLeft: user.conversionsLeft(),

});

} catch (error) {

console.error('Conversion error:', error);

res.status(500).json({ message: 'Conversion failed', error });

}

};

**3. Payment Processing**

javascript

Copy

Download

*// paymentController.js*

const Subscription = require('../models/Subscription');

const User = require('../models/User');

const { sendPaymentEmail } = require('../utils/email');

exports.processPayment = async (req, res) => {

try {

const { userId, plan } = req.body;

*// Validate plan*

const planDetails = getPlanDetails(plan);

if (!planDetails) {

return res.status(400).json({ message: 'Invalid plan' });

}

*// Create subscription record*

const subscription = new Subscription({

user: userId,

planType: plan,

amount: planDetails.price,

startDate: new Date(),

endDate: new Date(Date.now() + planDetails.duration \* 24 \* 60 \* 60 \* 1000),

});

await subscription.save();

*// Update user*

await User.findByIdAndUpdate(userId, {

$set: {

subscription: subscription.\_id,

trialUsed: true

}

});

*// Send payment confirmation (in reality, this would be after actual payment)*

await sendPaymentEmail(userId, subscription);

res.json({

success: true,

subscription,

message: 'Payment processed successfully'

});

} catch (error) {

res.status(500).json({ message: 'Payment processing failed', error });

}

};

function getPlanDetails(plan) {

const plans = {

monthly: { price: 15, duration: 30 },

'6month': { price: 70, duration: 180 },

yearly: { price: 120, duration: 365 },

};

return plans[plan];

}

**AI Integration**

**1. 2D-to-3D Conversion Service**

python

Copy

Download

*# ai\_conversion\_service.py*

import os

import cv2

import numpy as np

import tensorflow as tf

from flask import Flask, request, jsonify

from werkzeug.utils import secure\_filename

import uuid

app = Flask(\_\_name\_\_)

*# Load pre-trained model*

model = tf.keras.models.load\_model('2d\_to\_3d\_model.h5')

UPLOAD\_FOLDER = 'uploads'

OUTPUT\_FOLDER = 'outputs'

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

os.makedirs(OUTPUT\_FOLDER, exist\_ok=True)

def preprocess\_image(image\_path):

img = cv2.imread(image\_path)

img = cv2.resize(img, (256, 256))

img = img / 255.0

return np.expand\_dims(img, axis=0)

def convert\_to\_3d(image\_path):

*# Preprocess*

img\_array = preprocess\_image(image\_path)

*# Predict*

prediction = model.predict(img\_array)

*# Post-process to create 3D model file*

output\_filename = f"{str(uuid.uuid4())}.glb"

output\_path = os.path.join(OUTPUT\_FOLDER, output\_filename)

*# This would be replaced with actual conversion to 3D format*

*# For demo, we'll just save the processed image*

cv2.imwrite(output\_path, prediction[0] \* 255)

return output\_path

@app.route('/convert', methods=['POST'])

def convert():

if 'file' not in request.files:

return jsonify({"error": "No file uploaded"}), 400

file = request.files['file']

if file.filename == '':

return jsonify({"error": "No selected file"}), 400

filename = secure\_filename(file.filename)

upload\_path = os.path.join(UPLOAD\_FOLDER, filename)

file.save(upload\_path)

try:

output\_path = convert\_to\_3d(upload\_path)

return jsonify({

"success": True,

"model": output\_path

})

except Exception as e:

return jsonify({

"error": str(e)

}), 500

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

**2. Voice Command Processing**

python

Copy

Download

*# voice\_processor.py*

import openai

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

*# Configure OpenAI API*

openai.api\_key = os.getenv('OPENAI\_API\_KEY')

def process\_voice\_command(audio\_file):

*# Transcribe audio to text*

with open(audio\_file, 'rb') as f:

transcript = openai.Audio.transcribe("whisper-1", f)

*# Understand command using GPT*

response = openai.ChatCompletion.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are an AI assistant for Spatia, a 2D to 3D conversion app. Interpret user commands related to 3D modeling."},

{"role": "user", "content": transcript['text']}

]

)

return {

"transcript": transcript['text'],

"interpretation": response.choices[0].message.content,

"action": extract\_action(response.choices[0].message.content)

}

def extract\_action(text):

*# This would parse the GPT response to determine specific actions*

*# For demo, we'll return a simple structure*

return {

"type": "command",

"details": text

}

@app.route('/process-voice', methods=['POST'])

def handle\_voice():

if 'file' not in request.files:

return jsonify({"error": "No audio file"}), 400

file = request.files['file']

if file.filename == '':

return jsonify({"error": "No selected file"}), 400

try:

result = process\_voice\_command(file)

return jsonify(result)

except Exception as e:

return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5001)

**Deployment Strategy**

**1. Mobile App Deployment**

**iOS (App Store)**:

1. Create Apple Developer account ($99/year)
2. Set up app in App Store Connect
3. Configure app metadata, screenshots, and description
4. Build archive using Xcode
5. Submit for review

**Android (Play Store)**:

1. Create Google Play Developer account ($25 one-time)
2. Set up app in Play Console
3. Prepare store listing
4. Generate signed APK or App Bundle
5. Upload and publish

**2. Backend Deployment**

**Recommended**:

* AWS Elastic Beanstalk for Node.js backend
* AWS Lambda for AI services
* Amazon S3 for file storage
* MongoDB Atlas for database

**Alternative**:

* Heroku for simpler deployment
* Firebase for mobile-focused backend

**3. CI/CD Pipeline**

1. **GitHub Actions** for automated testing and deployment
2. **Fastlane** for automating app store deployments
3. **Monitoring** with Sentry for crash reporting
4. **Development**:
   * Mobile app development:
   * Backend development
   * AI integration:
5. **Infrastructure**
   * Backend hosting:
   * AI API costs:
   * Storage: 2
6. **Maintenance**:
   * Ongoing updates and bug fixes: 2,000−2,000−5,000/month

This comprehensive plan provides all the technical details needed to develop the Spatia mobile application with 2D-to-3D conversion capabilities, AI-powered features, and subscription-based monetization. The implementation can be adjusted based on specific requirements or budget constraints.