Technical Project Plan: Real-time Web Avatar

1. Objective

Create a single-page web application that uses a user's webcam to perform real-time facial expression tracking and applies those expressions to a 3D avatar loaded from a .glb file (e.g., from Ready Player Me). The application must be fully responsive and functional on mobile web browsers.

2. Core Technologies

- **3D Rendering:** three.js (for loading and displaying the 3D avatar).
- **Face Tracking:** @mediapipe/tasks-vision (specifically the FaceLandmarker task for detecting facial blend shapes).
- **Build Tool/Server:** vite (for a simple, fast local development server and module handling).
- **Avatar Model:** A .glb file with ARKit-compatible blend shapes (e.g., from Ready Player Me). Assume the file is named avatar.glb.

3. Project File Structure

```
web-animoji/
├── public/
│   ├── avatar.glb        # The 3D model
│   └── face_landmarker.task # The MediaPipe model file
├── index.html            # The main HTML file
├── style.css             # CSS for full-screen, mobile-first layout
└── main.js               # The core application logic
```

4. File-by-File Implementation Details

index.html

Objective: This file serves as the main entry point for the application.

- It must include a <meta name="viewport" ...> tag to ensure proper scaling and touch controls on mobile devices.
- It will contain a <canvas id="canvas"></canvas> element, which Three.js will use to render the 3D scene.
- It needs a hidden <video id="webcam" ...></video> element to stream the webcam feed for MediaPipe to process.

- It should include a <button id="startButton">Start</button> to get user permission for the webcam, which is required by browsers.
- It will link to style.css and import main.js as a module.

style.css

Objective: This file will make the application a full-screen, immersive experience.

- It must remove all default margins and padding from body and html and set their width and height to 100%.
- It should set overflow: hidden on the body to prevent scrolling.
- It will make the <canvas id="canvas"> element fill the entire screen (width: 100%, height: 100%).
- It will style the startButton and a loadingIndicator to be centered on the page.

main.js

Objective: This file contains all the core application logic to connect the webcam, 3D model, and face tracking.

- **Imports:** It will import THREE, GLTFLoader, OrbitControls from the 'three' library and FaceLandmarker from '@mediapipe/tasks-vision'.
- **Initialization:** On the startButton click, it will:
    1. Call initThreeJS(): Sets up the Three.js Scene, Camera, Renderer, and lights. It also adds OrbitControls for mobile touch interaction.
    2. Call initMediaPipe(): Initializes the FaceLandmarker, ensuring it's configured to output outputFaceBlendshapes: true.
    3. Call loadModel(): Uses GLTFLoader to load the avatar.glb file. It must traverse the loaded model to find the specific headMesh that contains the morphTargetDictionary.
    4. Call setupWebcam(): Gets the user's webcam stream (requesting facingMode: 'user') and links it to the hidden <video> element.
- **Animation Loop:** It must create a requestAnimationFrame loop (e.g., an animate() function) that runs every frame. Inside this loop, it will:
    1. Get the latest tracking data from faceLandmarker.detectForVideo().
    2. Check if valid blend shape data (results.faceBlendshapes) exists.
    3. If so, loop through the blendshapes array and map each shape.score to the corresponding headMesh.morphTargetInfluences[index].
    4. Update the OrbitControls.
    5. Call renderer.render(scene, camera) to draw the updated 3D avatar.

5. Required Model File

- Download the **face_landmarker.task** model file from MediaPipe's model page (approx. 24MB).
- Place this file in the public/ directory of your Vite project.