

Comprehensive Project Plan: Desktop 3D Viewer (P3DV)

This document outlines a complete plan for the P3DV application, including all discussed features and requirements. This plan is designed to be self-contained and ready for execution.

Project Name: Presentation 3D Viewer (P3DV)

Technology Stack:

- **Desktop Wrapper:** Electron
- **Frontend/UI:** React (with Vite for bundling)
- **3D Rendering:** react-three/fiber + three.js
- **Data/Storage:** SQLite3 (embedded metadata) + Local File System (GLB files)
- **Backend Logic:** Node.js (within Electron's Main Process)
- **Packaging:** electron-builder
- **Licensing:** Simple custom API for key validation.

Phase 1: Project Scaffolding & Core Architecture

The goal is to establish the project structure, install libraries, and configure the embedded SQLite database for offline-first operation.

Step	Detail/Goal	Deliverable Code/Instructions
1.1 Project Initialization	Set up a modern Electron + React project with a single-file structure for simplicity. Use Vite to handle the bundling.	npm create vite@latest p3dv-app -- --template react cd p3dv-app npm install electron electron-builder concurrently wait-on Update package.json with main process and builder configs.
1.2 Database Setup	Implement a local, file-based SQLite database. The database file will be created and	Install sqlite3 and better-sqlite3. Create a db/database.js file in the main process to

	managed within the application's data directory.	handle database initialization, schema creation, and basic CRUD operations.
1.3 3D Rendering Canvas	Integrate the core 3D libraries into the React frontend to confirm the rendering pipeline is functional.	Install three, @react-three/fiber, @react-three/drei. Create a basic App.jsx component with a <Canvas> and a simple mesh to test 3D rendering.
1.4 File System Manager	Create a utility to manage the application's local file storage for GLB files.	In the main process, create a utils/fs-manager.js file. This module should provide functions to get the application's data directory (app.getPath('userData')), save binary data to a file, and read files from that path.
1.5 Database Schema (Updated)	Design a robust database schema to support the new library features.	<p>models table:</p> <ul style="list-style-type: none"> - id (INTEGER, PRIMARY KEY, AUTOINCREMENT) - name (TEXT, NOT NULL) - local_path (TEXT, NOT NULL, UNIQUE) - category_id (INTEGER, FOREIGN KEY) - created_at (TEXT) <p>categories table:</p> <ul style="list-style-type: none"> - id (INTEGER, PRIMARY KEY, AUTOINCREMENT) - name (TEXT, NOT NULL, UNIQUE) <p>tags table (for custom flags):</p> <ul style="list-style-type: none"> - id (INTEGER, PRIMARY KEY, AUTOINCREMENT) - name (TEXT, NOT NULL,

		UNIQUE) model_tags table (junction table): - model_id (INTEGER, FOREIGN KEY) - tag_id (INTEGER, FOREIGN KEY)
--	--	---

Phase 2: Licensing & Offline Activation System

This phase addresses the client's requirement for a one-time activation key, ensuring the app is fully functional offline after validation.

Step	Detail/Goal	Deliverable Code/Instructions
2.1 Activation UI Component	Create a React component to display a simple activation form. This UI will be the first thing the user sees if the app is not activated.	Create a src/components/Activation Form.jsx with an input field for the license key and an "Activate" button. This component will call an IPC channel to send the key to the main process for validation.
2.2 Main Process Activation Logic	The main process will handle the entire activation workflow: check for an existing license, validate a new one via a remote API, and update the local SQLite database.	In main.js, implement a function to: <ol style="list-style-type: none"> 1. Check the SQLite is_activated field. 2. If false, display the Activation window. 3. Listen for the activate-key IPC event from the renderer. 4. Make a fetch() request to a remote API (https://your-licensing-api.com/validate-key) with the provided key. 5. On success, update the

		<p>SQLite is_activated field to true, then close the activation window.</p> <p>6. On failure, send an error message back to the renderer via IPC.</p>
2.3 Offline License Check	<p>Ensure the app can run offline after the initial activation.</p>	<p>Once is_activated is set to true in the SQLite database, the app should bypass the activation form on subsequent launches, even without an internet connection.</p>

Phase 3: Deep Linking & User Library Features

This phase implements the critical PowerPoint-to-App launch sequence and builds the user's model library UI and logic.

Step	Detail/Goal	Deliverable Code/Instructions
3.1 Single-Instance Logic	<p>Ensure only one instance of the application is ever running.</p>	<p>In main.js, use <code>app.requestSingleInstanceLock()</code>. If the lock fails, quit the second instance. Listen for the second-instance event to handle new URLs.</p>
3.2 Custom Protocol Registration	<p>Configure electron-builder to automatically register the <code>p3dv://</code> protocol during installation on both Windows and macOS.</p>	<p>Update package.json's build section with the protocols object.</p>
3.3 URL Handling & Parsing	<p>The main process will parse the incoming URL (<code>p3dv://view?id=...</code>) and extract the id of the model to display.</p>	<p>In main.js, set up a listener for the second-instance event (and open-url on macOS) to retrieve the URL from the command-line</p>

		arguments. Use <code>URLSearchParams</code> to get the <code>id</code> parameter.
3.4 User Library UI	Create a navigable UI with two main sections: Library and Community.	Use a simple routing approach in React. The Library will contain the user's models with filters for categories and tags. The Community section will be a placeholder with "Coming Soon" text.
3.5 Model Import Workflow	Implement an "Add Model" button that allows users to select a GLB file and fill out metadata.	Create a React component to handle file selection using an IPC call to the main process (<code>dialog.showOpenDialog</code>). The main process will: <ol style="list-style-type: none"> 1. Copy the selected GLB file to the app's data directory. 2. Store the new model's metadata (name, local path, category, tags) in the SQLite database.
3.6 Dynamic Deep Link Generation	For each model in the library, display a unique, copyable deep link based on its ID.	The models data in the React frontend will be mapped to a list. Each list item will display a button that, when clicked, copies the link (<code>p3dv://view?id=MODEL_ID</code>) to the clipboard using <code>document.execCommand('copy')</code> .
3.7 Filtering & Searching	Implement filtering logic for the library section based on categories and custom flags.	Create a simple search bar and filter dropdowns in the UI. The React component will query the SQLite

		database via IPC with WHERE clauses to fetch filtered results.
--	--	--

Phase 4: Build & Deployment

This final phase prepares the application for distribution, ensuring a seamless one-click installation for the end-user.

Step	Detail/Goal	Deliverable Code/Instructions
4.1 electron-builder Configuration	Configure the builder to create installers for Windows and macOS, including the automated protocol registration.	Ensure your package.json contains the protocols object under the build key. For Windows, configure NSIS to create a one-click installer. For macOS, configure the .dmg.
4.2 Test & Quality Assurance	Perform a final end-to-end test on both Windows and macOS to verify all features work as expected.	Test Cases: 1. Launch fresh install. Activation form appears. Enter key. App activates and proceeds. 2. Click p3dv://... link from PowerPoint. App launches, correct model loads. 3. Add a new model via the "Library" section, including categories and tags. 4. Test filtering and searching. 5. Click the "Return" button. App closes, PowerPoint regains focus. 6. Test offline functionality after activation. App should launch and function correctly.