30-Hour Hackathon Roadmap: AI + ZK Privacy on Cardano

1. The Tech Stack (Don't deviate from this)

- **Frontend:** Next.js (React) + Tailwind CSS.
- **Cardano SDK:** MeshJS (Handles wallet connection, transaction building, and minting).
- **Smart Contract:** Aiken (The only feasible choice for a 30h hackathon; compiles to Plutus V2).
- **AI Model:** Google Gemini API (Model: `gemini-3-pro` or "Nano Banana" specific endpoint if available publicly, otherwise standard Gemini Vision).
- **Storage:** Pinata SDK (for IPFS).
- **Wallet:** Nami or Eternal (Set to **Preprod Testnet**).
- **Faucet:** Cardano Testnet Faucet (to get free ADA).

2. Architecture Overview

1. **The Creator:** User prompts AI -> App calls Gemini (Nano Banana) -> Returns Image.
2. **The Mint:** App uploads Image to IPFS -> App mints NFT to a "Holding Contract" (not the user's wallet directly if you want privacy/vesting).
3. **The Privacy Layer:** The NFT metadata contains a "Hash" of a secret password.
4. **The ZK Proof:** To claim/transfer without revealing the link between the Creator and the Owner, the user submits a transaction proving they know the "Secret" that matches the "Hash" (This is a simplified ZK/Commit-Reveal scheme fitting for a hackathon).

3. Hour-by-Hour Roadmap

Phase 1: Setup & AI Generation (Hours 0-6)

Goal: A website where I can type a prompt and see an image.

- **Hour 0-1: Project Scaffold.**
  - Run `npx create-mesh-app` (Select Next.js, TypeScript, App Router).
  - Install generic dependencies: `npm install @pinata/sdk @google/generative-ai` .
- **Hour 1-2: Google "Nano Banana" Integration.**
  - Get a Gemini API Key from Google AI Studio.
  - Create a Next.js API route `/api/generate` .
  - **Snippet:**

```
const { GoogleGenerativeAI } = require("@google/generative-ai");
const genAI = new GoogleGenerativeAI(process.env.GOOGLE_API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-3-pro" }); // Or "nano-l
// Generate content...
```

- **Hour 3-6: IPFS Pipeline.**
  - User clicks "Mint".
  - Backend receives image buffer.
  - Upload to Pinata (`pinata.pinFileToIPFS`).
  - Construct the **CIP-25 Metadata** JSON object (standard NFT format).

Phase 2: The "Naive" Mint (Hours 6-12)

Goal: Mint the AI art as a standard NFT using MeshJS. (Get this working *before* adding ZK).

- **Hour 6-8: Wallet Connection.**
  - Use `<CardanoWallet />` component from MeshJS.
  - Ensure you are on **Preprod Testnet** and have funded your wallet via the Faucet.
- **Hour 8-12: The Minting Transaction.**
  - Use `MeshTxBuilder` or the `AssetForge` (if available in Mesh) to mint.
  - **Crucial:** You need a "Policy ID". MeshJS can generate a "Native Script" policy (simple time-lock) automatically.
  - *Deliverable:* You have an image in your Nami wallet.

Phase 3: The Smart Contract (Aiken) (Hours 12-20)

Goal: A contract that holds the NFT and only releases it if a "Proof" is provided.

- **Hour 12-14: Aiken Setup.**
  - Install `aiken` via cargo or binary.
  - `aiken new my-privacy-dapp` .
- **Hour 14-18: The Validator (Logic).**
  - **Concept:** Instead of a complex ZK-SNARK (Circom), use a **Hashed Timelock Contract (HTLC)** logic for the hackathon. It mimics ZK behavior (proving knowledge of a preimage without revealing the preimage until the claim).
  - **Logic:**
    - `datum` : The Hash of a secret (e.g., `sha2_256(secret)`).
    - `redeemer` : The actual `secret` .
    - **Validation:** `sha2_256(redeemer) == datum` .
- **Hour 18-20: Deploy & Test.**
  - Compile: `aiken build` .
  - Get the `plutus.json` .
  - Use `mesh` to deploy/interact with this script.

Phase 4: The "ZK" Privacy Layer (Hours 20-26)

Goal: Allow "Private Claiming".

- **The Hackathon Shortcut:**

- True ZK (Groth16/Plonk) on Cardano requires the BLS12-381 curves (available in Plutus V3/Chang, but hard to use).
- **Your Strategy:** The "Privacy" comes from the fact that the **Creator** generates the secret, hashes it, and mints the NFT to the **Contract Address** (not their own).
- The **Owner** (who knows the secret) can use a *fresh, anonymous wallet* to claim the NFT by providing the secret.
- *Result:* The blockchain sees `Contract -> Fresh Wallet` . It does NOT see `Creator Wallet -> Fresh Wallet` . You have achieved basic privacy/anonymity.

Phase 5: UI Polish & Pitch (Hours 26-30)

- **Hour 26-28: UI Cleanup.**
  - Add loading spinners (AI generation takes time).
  - Show "Success" modals with Cardanoscan links.
- **Hour 28-30: The Pitch.**
  - **Narrative:** "Current NFT privacy is nonexistent. We used a Commit-Reveal scheme on Aiken to decouple creation from ownership."
  - Record demo video.

4. Useful Snippets

MeshJS: Minting (Simplified)

```
import { MeshTxBuilder } from '@meshsdk/core';

const tx = new MeshTxBuilder({ setup: browserWallet });
await tx
  .mintPlutusScriptV2() // Or native mint
  .mint("1", policyId, tokenName)
  .metadata(metadata)
  .changeAddress(address)
  .complete();
```

Aiken: The "Secret" Validator

```
validator {
  fn spend(datum: ByteArray, redeemer: ByteArray, _ctx: ScriptContext) -> Bool {
    // Check if the hash of the provided secret (redeemer) matches the stored hash (dat
    sha2_256(redeemer) == datum
  }
}
```

5. Potential Pitfalls

1. **Plutus Versions:** Ensure MeshJS and Aiken are both targeting **Plutus V2**.

2. **Pinata Keys:** Don't expose them in the frontend. Use a Next.js API route (`pages/api/upload.ts`) to handle keys server-side.
3. **Browser Wallet:** Nami can be finicky with "Collateral". Make sure you set Collateral in Nami