Project layout

axiom-airdrop contracts RewardsVault.sol scripts deploy_vault.ts deposit.ts airdrop.ts readGrants.ts utils.ts data winners.csv .env.example hardhat.config.ts package.json tsconfig.json README.md

contracts RewardsVault.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

/*
AXM fixed supply airdrop vault
Holds pre-funded AXM and schedules time-locked airdrops
No minting, no token ownership required
*/

interface IERC20 {
    function transfer(address to, uint256 amt) external returns (bool);
    function transferFrom(address from, address to, uint256 amt) external returns (bool);
    function balanceOf(address who) external view returns (uint256);
    function approve(address spender, uint256 amt) external returns (bool);
}

contract RewardsVault {
    IERC20 public immutable AXM;
    address public owner;
    address public pendingOwner;

    uint256 public monthlyCap;
    mapping(uint256 => uint256) public distributedThisMonth;

    struct Grant {
        uint256 amount;
        uint64  unlockTime;
        bool    claimed;
    }
    mapping(address => Grant[]) public grants;

    event Deposit(uint256 amount);
    event AirdropScheduled(address indexed user, uint256 amount, uint64 unlockTime, uint256 grantIndex);
    event Claimed(address indexed user, uint256 amount, uint256 grantIndex);
    event OwnershipTransferStarted(address indexed newOwner);
    event OwnershipTransferred(address indexed newOwner);
    event MonthlyCapUpdated(uint256 newCap);

    modifier onlyOwner { require(msg.sender == owner, "not owner"); _; }

    constructor(address axm, uint256 initialMonthlyCap) {
        require(axm != address(0), "axm required");
        AXM = IERC20(axm);
        owner = msg.sender;
        monthlyCap = initialMonthlyCap;
    }

    function deposit(uint256 amount) external onlyOwner {
        require(AXM.transferFrom(msg.sender, address(this), amount), "transferFrom");
        emit Deposit(amount);
```

```
    }

    function monthIndex(uint256 ts) public pure returns (uint256) {
        return ts / 2592000;
    }

    function airdropBatch(address[] calldata users, uint256[] calldata amounts, uint64
lockDurationSeconds) external onlyOwner {
        require(users.length == amounts.length, "len mismatch");
        uint256 total;
        for (uint256 i = 0; i < amounts.length; i++) total += amounts[i];

        uint256 idx = monthIndex(block.timestamp);
        distributedThisMonth[idx] += total;
        require(distributedThisMonth[idx] <= monthlyCap, "monthly cap");

        for (uint256 i = 0; i < users.length; i++) {
            uint64 unlockTime = uint64(block.timestamp) + lockDurationSeconds;
            grants[users[i]].push(Grant({ amount: amounts[i], unlockTime: unlockTime, claimed: false
}));
            emit AirdropScheduled(users[i], amounts[i], unlockTime, grants[users[i]].length - 1);
        }
    }

    function claim(uint256 grantIndex) external {
        Grant storage g = grants[msg.sender][grantIndex];
        require(!g.claimed, "claimed");
        require(block.timestamp >= g.unlockTime, "locked");
        g.claimed = true;
        require(AXM.transfer(msg.sender, g.amount), "transfer");
        emit Claimed(msg.sender, g.amount, grantIndex);
    }

    function setMonthlyCap(uint256 newCap) external onlyOwner {
        monthlyCap = newCap;
        emit MonthlyCapUpdated(newCap);
    }

    function startOwnershipTransfer(address newOwner) external onlyOwner {
        pendingOwner = newOwner;
        emit OwnershipTransferStarted(newOwner);
    }
    function acceptOwnership() external {
        require(msg.sender == pendingOwner, "not pending");
        owner = pendingOwner;
        pendingOwner = address(0);
        emit OwnershipTransferred(owner);
    }
}
```

scripts utils.ts

```
import { config as dotenv } from "dotenv";
dotenv();

export const RPC_URL = process.env.RPC_URL as string;
export const ADMIN_PK = process.env.ADMIN_PK as string;
export const AXM_ADDR = process.env.AXM_ADDR as string;         // existing AXM token
```

```
export const VAULT_ADDR = process.env.VAULT_ADDR as string;     // set after deploy
export const MONTHLY_CAP_AXM = process.env.MONTHLY_CAP_AXM || "100000";
export const LOCK_SECONDS = process.env.LOCK_SECONDS || `${30 * 24 * 60 * 60}`;
```

scripts deploy_vault.ts

```
import { ethers } from "hardhat";
import { AXM_ADDR, MONTHLY_CAP_AXM } from "./utils";

async function main() {
  if (!AXM_ADDR) throw new Error("AXM_ADDR must be set in .env to your existing token");
  const [deployer] = await ethers.getSigners();
  console.log("deployer", deployer.address, "AXM", AXM_ADDR);

  const cap = ethers.parseUnits(MONTHLY_CAP_AXM, 18);
  const Vault = await ethers.getContractFactory("RewardsVault");
  const vault = await Vault.deploy(AXM_ADDR, cap);
  await vault.waitForDeployment();
  console.log("RewardsVault", await vault.getAddress());
}

main().catch((e) => { console.error(e); process.exit(1); });
```

scripts deposit.ts

```
import { ethers } from "hardhat";
import { AXM_ADDR, VAULT_ADDR } from "./utils";

async function main() {
  const amountHuman = process.argv[2] || "100000";
  const amount = ethers.parseUnits(amountHuman, 18);

  const [admin] = await ethers.getSigners();
  const axm = await ethers.getContractAt("IERC20", AXM_ADDR);
  const vault = await ethers.getContractAt("RewardsVault", VAULT_ADDR);

  const ap = await axm.approve(VAULT_ADDR, amount);
  await ap.wait();
  const tx = await vault.deposit(amount);
  await tx.wait();
  console.log("Deposited", amountHuman, "AXM to vault");
}

main().catch((e)=>{console.error(e);process.exit(1);});
```

scripts airdrop.ts

```
import { ethers } from "hardhat";
import fs from "fs";
import { VAULT_ADDR, LOCK_SECONDS } from "./utils";

function parseCsv(path: string) {
  const lines = fs.readFileSync(path, "utf8").trim().split("\n").slice(1);
  const users: string[] = [];
  const amts: bigint[] = [];
  for (const line of lines) {
    const parts = line.split(",");
```

```
    const wallet = parts[3].trim();
    const amt = ethers.parseUnits(parts[4].trim(), 18);
    users.push(wallet);
    amts.push(amt);
  }
  return { users, amts };
}

async function main() {
  const csv = process.argv[2] || "./data/winners.csv";
  const { users, amts } = parseCsv(csv);
  console.log("entries", users.length);

  const vault = await ethers.getContractAt("RewardsVault", VAULT_ADDR);
  const tx = await vault.airdropBatch(users, amts, Number(LOCK_SECONDS));
  console.log("tx", tx.hash);
  await tx.wait();
  console.log("airdrop scheduled");
}

main().catch((e)=>{console.error(e);process.exit(1);});
```

scripts readGrants.ts

```
import { ethers } from "hardhat";
import { VAULT_ADDR } from "./utils";

async function main() {
  const [me] = await ethers.getSigners();
  const vault = await ethers.getContractAt("RewardsVault", VAULT_ADDR);

  for (let i = 0; i < 50; i++) {
    try {
      const g = await vault.grants(me.address, i);
      console.log("grant", i, g.amount.toString(), g.unlockTime.toString(), g.claimed);
    } catch {
      break;
    }
  }
}

main().catch((e)=>{console.error(e);process.exit(1);});
```

data winners.csv

```
email,facebook_name,chain,wallet,amount_axm
alice@example.com,Alice Smith,peaq,0x1111111111111111111111111111111111111111,150
bob@example.com,Bob
Jones,polygon,0x2222222222222222222222222222222222222222,100
```

.env.example

```
RPC_URL=https://your-rpc
ADMIN_PK=0xprivatekey
AXM_ADDR=0xExistingAXMTokenAddress
VAULT_ADDR=0xDeployedVault
MONTHLY_CAP_AXM=100000
```

```
LOCK_SECONDS=2592000
```

hardhat.config.ts

```
import { HardhatUserConfig } from "hardhat/config";
import "@nomicfoundation/hardhat-toolbox";
import "@nomicfoundation/hardhat-ethers";
import "dotenv/config";

const config: HardhatUserConfig = {
  solidity: "0.8.24",
  networks: {
    custom: {
      url: process.env.RPC_URL || "",
      accounts: process.env.ADMIN_PK ? [process.env.ADMIN_PK] : []
    }
  }
};
export default config;
```

package.json

```
{
  "name": "axiom-airdrop",
  "version": "1.1.0",
  "private": true,
  "scripts": {
    "build": "hardhat compile",
    "deploy:vault": "hardhat run scripts/deploy_vault.ts --network custom",
    "deposit": "hardhat run scripts/deposit.ts --network custom",
    "airdrop": "hardhat run scripts/airdrop.ts --network custom",
    "grants": "hardhat run scripts/readGrants.ts --network custom"
  },
  "dependencies": {
    "@nomicfoundation/hardhat-ethers": "^3.0.5",
    "@nomicfoundation/hardhat-toolbox": "^5.0.0",
    "dotenv": "^16.4.5",
    "ethers": "^6.11.1",
    "hardhat": "^2.22.7",
    "ts-node": "^10.9.2",
    "typescript": "^5.6.3"
  }
}
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "outDir": "dist",
    "strict": true,
    "esModuleInterop": true,
    "resolveJsonModule": true,
    "types": ["node", "hardhat"]
  }
}
```

README.md

AXM airdrop system for fixed, non mintable 15B supply

Prereq
Set RPC_URL and ADMIN_PK in .env
Set AXM_ADDR to your existing token address

Install
yarn install
yarn build

Deploy vault
yarn deploy:vault
Copy printed RewardsVault address into .env as VAULT_ADDR

Fund vault from treasury
yarn deposit 100000
This approves and transfers AXM from your admin wallet into the vault

Weekly airdrop
Update data/winners.csv
yarn airdrop
Post transaction hash and leaderboard

Claims
Users call claim(grantIndex) after unlock
You can embed a simple claim button on your site using ethers v6

Replit AI build prompt

Begin build

Create a Hardhat TypeScript project named axiom-airdrop that uses an existing non mintable AXM token at address AXM_ADDR set in .env. Add the exact files and contents from my chat for RewardsVault.sol, scripts, data, configs, and package.json.

Install dependencies, compile, and create .env from .env.example. Then run yarn build.

Generate commands and sample output for: deploying RewardsVault with constructor args AXM_ADDR and a monthly cap read from MONTHLY_CAP_AXM approving and depositing AXM from the admin wallet into the vault running a weekly airdrop from data winners.csv using a 30 day lock reading back grants for the admin wallet

Finally, print a minimal web snippet using ethers v6 that allows a connected wallet to input a grant index and call claim on RewardsVault at VAULT_ADDR.

End build

Notes for your live token

Set AXM_ADDR to the deployed AXM address. Do vault funding only from your treasury or marketing wallet that already holds AXM. Monthly cap protects supply pacing. Adjust via setMonthlyCap if needed. Lock period is set per airdrop run. Default is 30 days in .env.

Install dependency

npm i ethers

Create ClaimGrants.tsx

```
import React, { useEffect, useMemo, useState } from "react";
import { BrowserProvider, Contract, formatUnits, parseUnits } from "ethers";

type Grant = {
  amount: string;
  unlockTime: string;
  claimed: boolean;
};

type Props = {
  vaultAddress: string;          // RewardsVault contract address
  axmDecimals?: number;          // defaults to 18
  rpcFallback?: string;          // optional read fallback RPC
  maxGrantsToScan?: number;      // safety upper bound, default 100
};

// Minimal ABI for RewardsVault
const VAULT_ABI = [
  "function grants(address,uint256) view returns (uint256 amount, uint64 unlockTime, bool claimed)",
  "function claim(uint256 grantIndex) external",
];

const humanTime = (unix: number) => {
  const d = new Date(unix * 1000);
  return `${d.toLocaleDateString()} ${d.toLocaleTimeString()}`;
};
```

```
const ClaimGrants: React.FC<Props> = ({
  vaultAddress,
  axmDecimals = 18,
  rpcFallback,
  maxGrantsToScan = 100,
}) => {
  const [provider, setProvider] = useState<BrowserProvider | null>(null);
  const [account, setAccount] = useState<string>("");
  const [grants, setGrants] = useState<Grant[]>([]);
  const [loading, setLoading] = useState<boolean>(false);
  const [claiming, setClaiming] = useState<number | null>(null);
  const [status, setStatus] = useState<string>("");

  const hasEthereum = typeof window !== "undefined" && (window as any).ethereum;

  const readProvider = useMemo(() => {
    if (provider) return provider;
    if (rpcFallback) {
      // Lazy import to avoid bundler node net polyfills when not used
      const { JsonRpcProvider } = require("ethers");
      return new JsonRpcProvider(rpcFallback);
    }
    return null;
  }, [provider, rpcFallback]);

  const connect = async () => {
    try {
      if (!hasEthereum) {
        setStatus("Wallet not found. Install a browser wallet.");
        return;
      }
      const p = new BrowserProvider((window as any).ethereum);
      const accounts = await p.send("eth_requestAccounts", []);
```

```
    setProvider(p);
    setAccount(accounts[0] || "");
    setStatus("Connected");
  } catch (e: any) {
    setStatus(e.message || "Connection failed");
  }
};

const fetchGrants = async () => {
  if (!readProvider || !account) return;
  setLoading(true);
  try {
    const vault = new Contract(vaultAddress, VAULT_ABI, readProvider);
    const out: Grant[] = [];
    for (let i = 0; i < maxGrantsToScan; i++) {
      try {
        const g = await vault.grants(account, i);
        const amount = formatUnits(g.amount, axmDecimals);
        const unlockTime = g.unlockTime.toString();
        out.push({
          amount,
          unlockTime,
          claimed: g.claimed,
        });
      } catch {
        // stop when grants throws for missing index
        break;
      }
    }
    setGrants(out);
    setStatus(`Found ${out.length} grants`);
  } catch (e: any) {
    setStatus(e.message || "Read failed");
```

```
  } finally {
    setLoading(false);
  }
};

const claim = async (index: number) => {
  if (!provider) {
    setStatus("Connect wallet first");
    return;
  }
  setClaiming(index);
  try {
    const signer = await provider.getSigner();
    const vault = new Contract(vaultAddress, VAULT_ABI, signer);
    const tx = await vault.claim(index);
    setStatus(`Claim sent. Tx ${tx.hash}`);
    await tx.wait();
    setStatus("Claim confirmed");
    await fetchGrants();
  } catch (e: any) {
    setStatus(e.info?.error?.message || e.message || "Claim failed");
  } finally {
    setClaiming(null);
  }
};

useEffect(() => {
  // Auto connect if wallet already authorized
  (async () => {
    if (hasEthereum && !provider) {
      const p = new BrowserProvider((window as any).ethereum);
      try {
        const accounts = await p.send("eth_accounts", []);
```

```
        if (accounts && accounts[0]) {
          setProvider(p);
          setAccount(accounts[0]);
        }
      } catch {}
    }
  })();
}, [hasEthereum, provider]);

useEffect(() => {
  if (account) fetchGrants();
}, [account]); // eslint-disable-line

return (
  <div className="p-4 max-w-2xl mx-auto">
    <div className="flex items-center justify-between mb-3">
      <div>
        <div className="text-lg font-semibold">AXM RewardsVault Claims</div>
        <div className="text-sm opacity-70">
          Vault {vaultAddress.slice(0, 10)}...{vaultAddress.slice(-6)}
        </div>
      </div>
      {!account ? (
        <button
          onClick={connect}
          className="px-4 py-2 rounded-lg bg-black text-white"
        >
          Connect Wallet
        </button>
      ) : (
        <div className="text-sm break-all max-w-xs text-right">
          {account}
        </div>
```

```
  )}
</div>

<div className="text-sm mb-3 opacity-80">
  Status: {status || "Idle"}
</div>

<div className="mb-4">
  <button
    onClick={fetchGrants}
    disabled={!account || loading}
    className="px-3 py-2 rounded-lg border"
  >
    Refresh Grants
  </button>
</div>

{loading ? (
  <div>Loading grants</div>
) : grants.length === 0 ? (
  <div>No grants found for this wallet</div>
) : (
  <div className="space-y-3">
    {grants.map((g, i) => {
      const now = Math.floor(Date.now() / 1000);
      const unlocked = Number(g.unlockTime) <= now;
      return (
        <div key={i} className="border rounded-lg p-3">
          <div className="font-medium">Grant {i}</div>
          <div className="text-sm">
            Amount {g.amount} AXM
          </div>
          <div className="text-sm">
```

```
                Unlocks {humanTime(Number(g.unlockTime))}
              </div>
              <div className="text-sm">
                Claimed {g.claimed ? "Yes" : "No"}
              </div>
              <div className="mt-2">
                <button
                  onClick={() => claim(i)}
                  disabled={!unlocked || g.claimed || claiming === i}
                  className={`px-3 py-2 rounded-lg ${
                    unlocked && !g.claimed && claiming !== i
                      ? "bg-black text-white"
                      : "bg-gray-200 text-gray-600"
                  }`}
                >
                  {claiming === i ? "Claiming" : "Claim"}
                </button>
              </div>
            </div>
          );
        })}
      </div>
    )}
  </div>
);
};

export default ClaimGrants;
```

How to use inside your React app

1. Place ClaimGrants.tsx anywhere under src.

2. In your page or route component:

```
import React from "react";
import ClaimGrants from "./ClaimGrants";

export default function ClaimsPage() {
  return (
    <ClaimGrants
      vaultAddress="0xYourRewardsVaultAddress"
      axmDecimals={18}
      rpcFallback="https://your-read-rpc"
      maxGrantsToScan={100}
    />
  );
}
```

Notes

Works with any EVM chain where your vault is deployed.

rpcFallback is optional but useful for read calls when the wallet is not connected.

No separate HTML entry needed. Pure React with JSX and Tailwind-friendly classNames.