**SORA AI FINANCIAL DAO — WHITEPAPER (v0.1)**

**1. INTRODUCTION**

**SORA AI Financial DAO** is a decentralized investment ecosystem where artificial intelligence (AI) autonomously analyzes markets, generates investment strategies, and a decentralized autonomous organization (DAO) governs, approves, and executes those strategies. Our goal is to offer a transparent, permissionless, and community-driven financial platform — where AI does the thinking and the community does the deciding.

**2. MISSION & VISION**

* **Mission:** Democratize intelligent investing by fusing AI analytics with decentralized governance, enabling anyone to participate in future-ready asset management.
* **Vision:** A global, borderless, and self-improving financial cooperative — governed by the crowd, powered by AI, and aligned with real-time data and decentralized logic.

**3. CORE COMPONENTS**

**3.1 AI Strategy Engine**

* Collects real-time market data from DEXs, CEXs, news, social sentiment, and macro indicators.
* Uses machine learning models (reinforcement learning, neural nets, sentiment NLP) to propose weekly portfolio strategies.
* Example Output: “This week’s strategy: ETH 50%, LINK 25%, DAI 25%.”

**3.2 DAO Governance System**

* Built on-chain using Aragon or custom Solidity contracts.
* Token holders ($SORA) vote on AI-proposed strategies.
* Includes mechanisms for proposal, discussion, voting, and implementation.

**3.3 Smart Contract Vault**

* Executes voted strategies via automated DeFi interactions (e.g. Uniswap, Aave, Lido).
* Tracks profits and rebalances portfolios.
* Non-custodial: Users maintain ownership; assets are governed via contracts.

**3.4 Frontend Interface (DApp)**

* Connect with MetaMask, WalletConnect, etc.
* Users can: stake funds, vote on strategies, review analytics, and receive performance-based rewards.

**4. TOKENOMICS**

* Token: **$SORA** (ERC-20)
* Total Supply: 100,000,000

| **Category** | **Allocation** | **Vesting Schedule** |
| --- | --- | --- |
| DAO Treasury | 40% | 4-year linear vesting |
| Community Rewards | 25% | Based on activity & loyalty |
| Team & Advisors | 20% | 1-year cliff, 3-year vesting |
| Strategic Partners | 10% | 6-month lock, 2-year vesting |
| Liquidity Pool | 5% | Initial listing + reserves |

* Utility:
  + Governance voting
  + Strategy proposal rights
  + Fee discounts and staking rewards

**5. GOVERNANCE FLOW**

1. AI publishes a new weekly portfolio strategy (auto-posted to dashboard).
2. $SORA holders review and vote on it via smart contract UI.
3. If approved (e.g. 60% quorum, 51% majority), strategy is auto-executed.
4. Smart contracts reallocate funds according to the plan.
5. After 7 days, performance is reviewed, profits (if any) are shared proportionally.
6. Community can also override AI with alternative human-proposed strategies.

**6. SECURITY & TRANSPARENCY**

* AI model weights and data sources published openly.
* Every contract is open source and verifiable.
* DAO decisions immutably recorded on-chain.
* Routine audits and bug bounty programs planned.

**7. ROADMAP SNAPSHOT**

| **Phase** | **Timeline** | **Milestones** |
| --- | --- | --- |
| Phase 1 | Q2 2025 | MVP: AI Engine + Voting UI on testnet |
| Phase 2 | Q3 2025 | Token launch, DAO deployment, user onboarding |
| Phase 3 | Q4 2025 | Multi-chain expansion, AI optimization, revenue-sharing DApp |
| Phase 4 | 2026+ | Full decentralization, community-led fund governance, advanced AI agents |

**8. CONCLUSION**

SORA AI Financial DAO is more than just an investment platform — it’s a decentralized movement toward intelligent finance. With AI at the core and the crowd in control, we aim to become the blueprint for the next generation of borderless, automated, and community-owned financial systems.

**"Let the AI invest. Let the DAO decide."**

**SORA AI Financial DAO – Tokenomics Model (v0.1)**

**1. Token Overview**

* **Token Name:** SORA Token
* **Ticker Symbol:** $SORA
* **Standard:** ERC-20
* **Total Supply:** 100,000,000 $SORA (fixed)

**2. Allocation Breakdown**

| **Category** | **Allocation** | **Tokens** | **Vesting Details** |
| --- | --- | --- | --- |
| DAO Treasury | 40% | 40,000,000 | 4-year linear vesting |
| Community Rewards | 25% | 25,000,000 | Distributed based on activity & participation |
| Team & Advisors | 20% | 20,000,000 | 1-year cliff, then 3-year linear vesting |
| Strategic Partners | 10% | 10,000,000 | 6-month lockup, 2-year vesting |
| Liquidity Provision | 5% | 5,000,000 | Initial DEX/launchpad + reserves |

**3. Utility & Incentives**

* **Governance:**
  + Vote on AI strategy proposals
  + Vote on fund allocation or protocol upgrades
* **Staking & Rewards:**
  + Stake $SORA to earn protocol fees & rewards
  + Boost voting weight via delegated staking
* **Proposal Power:**
  + Minimum threshold of tokens required to submit alternative investment proposals
* **Fee Discounts:**
  + $SORA holders receive reduced protocol fees

**4. Inflation/Deflation Policy**

* No inflation – fixed supply model
* Long-term deflation pressure via:
  + Buybacks (using protocol profits)
  + Governance-voted burns (optional)

**5. Launch & Liquidity Plan**

* **Initial Launch:**
  + Token generation event (TGE) in Q3 2025
  + 5% allocated to initial liquidity (DEX)
* **Liquidity Mining:**
  + Incentives for early LPs during first 12 weeks
  + Rewards issued from Community bucket
* **Staking Pools:**
  + Phase 1: $SORA single staking pool
  + Phase 2: ETH/SORA or USDC/SORA LP staking options

**6. Token Valuation Targets (for simulation purposes)**

| **Scenario** | **Circulating Supply** | **Market Cap Goal** | **Token Price** |
| --- | --- | --- | --- |
| Conservative | 20M | $10M | $0.50 |
| Base Case | 40M | $60M | $1.50 |
| Bullish | 60M | $180M | $3.00 |

**7. Treasury Use of Funds**

| **Category** | **% of Treasury** | **Use Case** |
| --- | --- | --- |
| AI R&D | 25% | Model training, datasets |
| DAO Ops & Grants | 25% | Developer bounties, incentives |
| Marketing | 20% | Community building, awareness |
| Legal & Security | 15% | Smart contract audits, DAO legal |
| Protocol Reserves | 15% | Emergency / growth buffer |

This model can be adjusted for real-world simulation using Google Sheets or Excel. Next step: dashboard mockup + live analytics chart for token emissions & DAO participation.

## import yfinance as yf

**import pandas as pd**

**import numpy as np**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.model\_selection import train\_test\_split**

**from datetime import datetime, timedelta**

**# ------------------**

**# CONFIGURATION**

**# ------------------**

**assets = ["ETH-USD", "BTC-USD", "SOL-USD", "LINK-USD"]**

**lookback\_days = 90**

**def fetch\_data(symbol, days):**

**end = datetime.today()**

**start = end - timedelta(days=days)**

**df = yf.download(symbol, start=start, end=end)**

**df["Symbol"] = symbol**

**return df**

**# ------------------**

**# DATA COLLECTION**

**# ------------------**

**all\_data = [fetch\_data(asset, lookback\_days) for asset in assets]**

**data = pd.concat(all\_data)**

**data.reset\_index(inplace=True)**

**# ------------------**

**# FEATURE ENGINEERING**

**# ------------------**

**data["Return"] = data["Adj Close"].pct\_change()**

**data["Volatility"] = data["Return"].rolling(window=7).std()**

**data["Momentum"] = data["Adj Close"] / data["Adj Close"].shift(7) - 1**

**data.dropna(inplace=True)**

**# ------------------**

**# LABEL GENERATION**

**# ------------------**

**data["Target"] = (data["Return"].shift(-1) > 0).astype(int)**

**# ------------------**

**# MODEL TRAINING & PREDICTION**

**# ------------------**

**features = ["Return", "Volatility", "Momentum"]**

**strategy\_output = {}**

**for symbol in assets:**

**df\_asset = data[data.Symbol == symbol]**

**X = df\_asset[features]**

**y = df\_asset["Target"]**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)**

**model = RandomForestClassifier(n\_estimators=100, random\_state=42)**

**model.fit(X\_train, y\_train)**

**# Predict tomorrow**

**latest = X.iloc[-1].values.reshape(1, -1)**

**prob = model.predict\_proba(latest)[0][1] # probability of positive return**

**strategy\_output[symbol] = prob**

**# ------------------**

**# STRATEGY WEIGHTING**

**# ------------------**

**total = sum(strategy\_output.values())**

**weights = {k: round(v / total, 2) for k, v in strategy\_output.items()}**

**print("\n📈 Suggested Portfolio Allocation (AI-Driven):")**

**for asset, weight in weights.items():**

**print(f"- {asset.replace('-USD','')}: {weight \* 100:.0f}%")**

**import yfinance as yf**

**import pandas as pd**

**import numpy as np**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.model\_selection import train\_test\_split**

**from datetime import datetime, timedelta**

**from fastapi import FastAPI**

**from pydantic import BaseModel**

**import uvicorn**

**app = FastAPI()**

**assets = ["ETH-USD", "BTC-USD", "SOL-USD", "LINK-USD"]**

**lookback\_days = 90**

**class StrategyOutput(BaseModel):**

**asset: str**

**weight: float**

**@app.get("/suggested-allocation")**

**def get\_suggested\_allocation():**

**def fetch\_data(symbol, days):**

**end = datetime.today()**

**start = end - timedelta(days=days)**

**df = yf.download(symbol, start=start, end=end)**

**df["Symbol"] = symbol**

**return df**

**all\_data = [fetch\_data(asset, lookback\_days) for asset in assets]**

**data = pd.concat(all\_data)**

**data.reset\_index(inplace=True)**

**data["Return"] = data["Adj Close"].pct\_change()**

**data["Volatility"] = data["Return"].rolling(window=7).std()**

**data["Momentum"] = data["Adj Close"] / data["Adj Close"].shift(7) - 1**

**data.dropna(inplace=True)**

**data["Target"] = (data["Return"].shift(-1) > 0).astype(int)**

**features = ["Return", "Volatility", "Momentum"]**

**strategy\_output = {}**

**for symbol in assets:**

**df\_asset = data[data.Symbol == symbol]**

**X = df\_asset[features]**

**y = df\_asset["Target"]**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)**

**model = RandomForestClassifier(n\_estimators=100, random\_state=42)**

**model.fit(X\_train, y\_train)**

**latest = X.iloc[-1].values.reshape(1, -1)**

**prob = model.predict\_proba(latest)[0][1]**

**strategy\_output[symbol] = prob**

**total = sum(strategy\_output.values())**

**weights = {k: round(v / total, 2) for k, v in strategy\_output.items()}**

**response = [StrategyOutput(asset=k.replace("-USD", ""), weight=w) for k, w in weights.items()]**

**return response**

**if \_\_name\_\_ == "\_\_main\_\_":**

**uvicorn.run(app, host="0.0.0.0", port=8000)**

**import React from "react";**

**import { Card, CardContent } from "@/components/ui/card";**

**import { Button } from "@/components/ui/button";**

**import { Progress } from "@/components/ui/progress";**

**import { Vote } from "lucide-react";**

**const mockProposals = [**

**{**

**id: 1,**

**title: "Rebalance Portfolio to: 40% ETH, 30% BTC, 20% SOL, 10% LINK",**

**description: "Suggested by AI Strategy Engine on April 18, 2025.",**

**votesFor: 65,**

**votesAgainst: 35,**

**deadline: "2025-04-20"**

**},**

**{**

**id: 2,**

**title: "Allocate 10% Treasury to AI model training",**

**description: "Proposed by core team. Estimated cost: $100K.",**

**votesFor: 120,**

**votesAgainst: 22,**

**deadline: "2025-04-25"**

**}**

**];**

**export default function DAOProposalBoard() {**

**return (**

**<div className="grid grid-cols-1 gap-6 p-6">**

**{mockProposals.map((proposal) => (**

**<Card key={proposal.id} className="rounded-2xl shadow-lg">**

**<CardContent className="p-6">**

**<h2 className="text-xl font-bold mb-2">{proposal.title}</h2>**

**<p className="text-sm text-gray-500 mb-4">{proposal.description}</p>**

**<div className="mb-2 text-sm">Voting ends: {proposal.deadline}</div>**

**<div className="flex items-center gap-4 mb-4">**

**<div className="w-full">**

**<Progress value={proposal.votesFor} className="h-2 bg-green-500" />**

**<span className="text-xs text-green-600">For: {proposal.votesFor} votes</span>**

**</div>**

**<div className="w-full">**

**<Progress value={proposal.votesAgainst} className="h-2 bg-red-500" />**

**<span className="text-xs text-red-600">Against: {proposal.votesAgainst} votes</span>**

**</div>**

**</div>**

**<div className="flex gap-4">**

**<Button variant="default" className="w-full">**

**<Vote className="mr-2 h-4 w-4" /> Vote For**

**</Button>**

**<Button variant="destructive" className="w-full">**

**<Vote className="mr-2 h-4 w-4 rotate-180" /> Vote Against**

**</Button>**

**</div>**

**</CardContent>**

**</Card>**

**))}**

**</div>**

**);**

**}**

**// SPDX-License-Identifier: MIT**

**pragma solidity ^0.8.19;**

**contract SoraDAO {**

**struct Proposal {**

**uint id;**

**string title;**

**string description;**

**uint votesFor;**

**uint votesAgainst;**

**uint deadline;**

**bool executed;**

**}**

**uint public nextProposalId;**

**mapping(uint => Proposal) public proposals;**

**mapping(uint => mapping(address => bool)) public hasVoted;**

**address public owner;**

**uint public votingPeriod = 3 days;**

**event ProposalCreated(uint id, string title);**

**event Voted(uint proposalId, address voter, bool support);**

**event ProposalExecuted(uint id);**

**modifier onlyOwner() {**

**require(msg.sender == owner, "Not owner");**

**\_;**

**}**

**constructor() {**

**owner = msg.sender;**

**}**

**function createProposal(string memory \_title, string memory \_description) external onlyOwner {**

**proposals[nextProposalId] = Proposal({**

**id: nextProposalId,**

**title: \_title,**

**description: \_description,**

**votesFor: 0,**

**votesAgainst: 0,**

**deadline: block.timestamp + votingPeriod,**

**executed: false**

**});**

**emit ProposalCreated(nextProposalId, \_title);**

**nextProposalId++;**

**}**

**function vote(uint proposalId, bool support) external {**

**Proposal storage proposal = proposals[proposalId];**

**require(block.timestamp < proposal.deadline, "Voting ended");**

**require(!hasVoted[proposalId][msg.sender], "Already voted");**

**if (support) {**

**proposal.votesFor++;**

**} else {**

**proposal.votesAgainst++;**

**}**

**hasVoted[proposalId][msg.sender] = true;**

**emit Voted(proposalId, msg.sender, support);**

**}**

**function executeProposal(uint proposalId) external onlyOwner {**

**Proposal storage proposal = proposals[proposalId];**

**require(block.timestamp >= proposal.deadline, "Voting not ended");**

**require(!proposal.executed, "Already executed");**

**proposal.executed = true;**

**emit ProposalExecuted(proposalId);**

**}**

**}**

**import React, { useState, useEffect } from "react";**

**import { ethers } from "ethers";**

**import { Button } from "@/components/ui/button";**

**import SoraDAOABI from "@/contracts/SoraDAO.json";**

**const contractAddress = "0xYourContractAddressHere"; // Replace with actual deployed contract**

**export default function VoteInterface() {**

**const [provider, setProvider] = useState(null);**

**const [signer, setSigner] = useState(null);**

**const [contract, setContract] = useState(null);**

**const [account, setAccount] = useState("");**

**const [proposals, setProposals] = useState([]);**

**useEffect(() => {**

**const load = async () => {**

**if (window.ethereum) {**

**const tempProvider = new ethers.providers.Web3Provider(window.ethereum);**

**const tempSigner = tempProvider.getSigner();**

**const tempContract = new ethers.Contract(contractAddress, SoraDAOABI.abi, tempSigner);**

**const accounts = await window.ethereum.request({ method: "eth\_requestAccounts" });**

**setProvider(tempProvider);**

**setSigner(tempSigner);**

**setContract(tempContract);**

**setAccount(accounts[0]);**

**const count = await tempContract.nextProposalId();**

**const fetched = [];**

**for (let i = 0; i < count; i++) {**

**const proposal = await tempContract.proposals(i);**

**fetched.push(proposal);**

**}**

**setProposals(fetched);**

**}**

**};**

**load();**

**}, []);**

**const vote = async (id, support) => {**

**try {**

**const tx = await contract.vote(id, support);**

**await tx.wait();**

**alert("Vote submitted successfully.");**

**} catch (err) {**

**console.error(err);**

**alert("Voting failed.");**

**}**

**};**

**return (**

**<div className="p-6 space-y-4">**

**<h1 className="text-2xl font-bold mb-4">SORA DAO Voting Interface</h1>**

**<div className="text-sm text-gray-500">Connected: {account}</div>**

**{proposals.map((p, idx) => (**

**<div key={idx} className="border p-4 rounded-xl">**

**<div className="font-semibold">{p.title}</div>**

**<div className="text-sm text-gray-500 mb-2">{p.description}</div>**

**<div className="flex gap-2">**

**<Button onClick={() => vote(p.id, true)}>Vote For</Button>**

**<Button onClick={() => vote(p.id, false)} variant="destructive">**

**Vote Against**

**</Button>**

**</div>**

**</div>**

**))}**

**</div>**

**);**

**}**

**import React, { useState, useEffect } from "react";**

**import { ethers } from "ethers";**

**import { Button } from "@/components/ui/button";**

**import SoraDAOABI from "@/contracts/SoraDAO.json";**

**const contractAddress = "0x30e43D59E3346c407539A00b332541100C78909f";**

**export default function VoteInterface() {**

**const [provider, setProvider] = useState(null);**

**const [signer, setSigner] = useState(null);**

**const [contract, setContract] = useState(null);**

**const [account, setAccount] = useState("");**

**const [proposals, setProposals] = useState([]);**

**const [newTitle, setNewTitle] = useState("");**

**const [newDescription, setNewDescription] = useState("");**

**useEffect(() => {**

**const load = async () => {**

**if (window.ethereum) {**

**const tempProvider = new ethers.providers.Web3Provider(window.ethereum);**

**const tempSigner = tempProvider.getSigner();**

**const tempContract = new ethers.Contract(contractAddress, SoraDAOABI.abi, tempSigner);**

**const accounts = await window.ethereum.request({ method: "eth\_requestAccounts" });**

**setProvider(tempProvider);**

**setSigner(tempSigner);**

**setContract(tempContract);**

**setAccount(accounts[0]);**

**const count = await tempContract.nextProposalId();**

**const fetched = [];**

**for (let i = 0; i < count; i++) {**

**const proposal = await tempContract.proposals(i);**

**fetched.push(proposal);**

**}**

**setProposals(fetched);**

**}**

**};**

**load();**

**}, []);**

**const vote = async (id, support) => {**

**try {**

**const tx = await contract.vote(id, support);**

**await tx.wait();**

**alert("Vote submitted successfully.");**

**} catch (err) {**

**console.error(err);**

**alert("Voting failed.");**

**}**

**};**

**const createProposal = async () => {**

**try {**

**const tx = await contract.createProposal(newTitle, newDescription);**

**await tx.wait();**

**alert("Proposal cr**

**import React, { useState, useEffect } from "react";**

**import { ethers } from "ethers";**

**import { Button } from "@/components/ui/button";**

**import SoraDAOABI from "@/contracts/SoraDAO.json";**

**const contractAddress = "0x30e43D59E3346c407539A00b332541100C78909f";**

**export default function VoteInterface() {**

**const [provider, setProvider] = useState(null);**

**const [signer, setSigner] = useState(null);**

**const [contract, setContract] = useState(null);**

**const [account, setAccount] = useState("");**

**const [proposals, setProposals] = useState([]);**

**const [newTitle, setNewTitle] = useState("");**

**const [newDescription, setNewDescription] = useState("");**

**useEffect(() => {**

**const load = async () => {**

**if (window.ethereum) {**

**const tempProvider = new ethers.providers.Web3Provider(window.ethereum);**

**const tempSigner = tempProvider.getSigner();**

**const tempContract = new ethers.Contract(contractAddress, SoraDAOABI.abi, tempSigner);**

**const accounts = await window.ethereum.request({ method: "eth\_requestAccounts" });**

**setProvider(tempProvider);**

**setSigner(tempSigner);**

**setContract(tempContract);**

**setAccount(accounts[0]);**

**const count = await tempContract.nextProposalId();**

**const fetched = [];**

**for (let i = 0; i < count; i++) {**

**const proposal = await tempContract.proposals(i);**

**fetched.push(proposal);**

**}**

**setProposals(fetched);**

**}**

**};**

**load();**

**}, []);**

**const vote = async (id, support) => {**

**try {**

**const tx = await contract.vote(id, support);**

**await tx.wait();**

**alert("Vote submitted successfully.");**

**} catch (err) {**

**console.error(err);**

**alert("Voting failed.");**

**}**

**};**

**const createProposal = async () => {**

**try {**

**const tx = await contract.createProposal(newTitle, newDescription);**

**await tx.wait();**

**alert("Proposal created successfully.");**

**setNewTit**

**import React, { useState, useEffect } from "react";**

**import { ethers } from "ethers";**

**import { Button } from "@/components/ui/button";**

**import SoraDAOABI from "@/contracts/SoraDAO.json";**

**const contractAddress = process.env.NEXT\_PUBLIC\_CONTRACT\_ADDRESS || "0x30e43D59E3346c407539A00b332541100C78909f";**

**export default function VoteInterface() {**

**const [provider, setProvider] = useState(null);**

**const [signer, setSigner] = useState(null);**

**const [contract, setContract] = useState(null);**

**const [account, setAccount] = useState("");**

**const [proposals, setProposals] = useState([]);**

**const [newTitle, setNewTitle] = useState("");**

**const [newDescription, setNewDescription] = useState("");**

**useEffect(() => {**

**const load = async () => {**

**if (window.ethereum) {**

**const tempProvider = new ethers.providers.Web3Provider(window.ethereum);**

**const tempSigner = tempProvider.getSigner();**

**const tempContract = new ethers.Contract(contractAddress, SoraDAOABI.abi, tempSigner);**

**const accounts = await window.ethereum.request({ method: "eth\_requestAccounts" });**

**setProvider(tempProvider);**

**setSigner(tempSigner);**

**setContract(tempContract);**

**setAccount(accounts[0]);**

**const count = await tempContract.nextProposalId();**

**const fetched = [];**

**for (let i = 0; i < count; i++) {**

**const proposal = await tempContract.proposals(i);**

**fetched.push(proposal);**

**}**

**setProposals(fetched);**

**}**

**};**

**load();**

**}, []);**

**const vote = async (id, support) => {**

**try {**

**const tx = await contract.vote(id, support);**

**await tx.wait();**

**alert("Vote submitted successfully.");**

**} catch (err) {**

**console.error(err);**

**alert("Voting failed.");**

**}**

**};**

**const createProposal = async () => {**

**try {**

**const tx = await contract.createProposal(newTitle, newDescription);**

**await tx.wait();**

**alert("Proposal created successfully.");**

**setNewTitle("");**

**setNewDescription("");**

**} catch (err) {**

**console.error(err);**

**alert("Proposal creation failed.");**

**}**

**};**

**const executeProposal = async (id) => {**

**try {**

**const tx = await contract.executeProposal(id);**

**await tx.wait();**

**alert("Proposal executed successfully.");**

**} catch (err) {**

**console.error(err);**

**alert("Execution failed.");**

**}**

**};**

**return (**

**<div className="p-6 space-y-4">**

**<h1 className="text-2xl font-bold mb-4">SORA DAO Voting Interface</h1>**

**<div className="text-sm text-gray-500">Connected: {account}</div>**

**<div className="border p-4 rounded-xl space-y-2">**

**<h2 className="text-lg font-semibold">Create Proposal</h2>**

**<input**

**type="text"**

**placeholder="Title"**

**value={newTitle}**

**onChange={(e) => setNewTitle(e.target.value)}**

**className="w-full border rounded p-2"**

**/>**

**<textarea**

**placeholder="Description"**

**value={newDescription}**

**onChange={(e) => setNewDescription(e.target.value)}**

**className="w-full border rounded p-2"**

**/>**

**<Button onClick={createProposal}>Submit Proposal</Button>**

**</div>**

**{proposals.map((p, idx) => (**

**<div key={idx} className="border p-4 rounded-xl">**

**<div className="font-semibold">{p.title}</div>**

**<div className="text-sm text-gray-500 mb-2">{p.description}</div>**

**<div className="flex gap-2 mb-2">**

**<Button onClick={() => vote(p.id, true)}>Vote For</Button>**

**<Button onClick={() => vote(p.id, false)} variant="destructive">**

**Vote Against**

**</Button>**

**</div>**

**<Button onClick={() => executeProposal(p.id)} variant="outline">**

**Execute Proposal**

**</Button>**

**</div>**

**))}**

**</div>**

**);**

**}**

**📁 \*\*SORA AI Financial DAO - GitHub Structure\*\***

**```**

**/sora-dao**

**├── /app**

**│ └── page.tsx # Main voting UI (from VoteInterface)**

**├── /components**

**│ └── ui**

**│ └── button.tsx # UI button component (e.g. shadcn/ui or custom)**

**├── /contracts**

**│ └── SoraDAO.json # ABI of your deployed DAO contract**

**├── .env.local # Environment variables**

**├── tailwind.config.js # Tailwind CSS config**

**├── postcss.config.js # PostCSS for Tailwind**

**├── tsconfig.json # TypeScript config**

**├── package.json # Project dependencies**

**├── README.md # Project overview**

**```**

**---**

**📦 \*\*1. `page.tsx` (UI logic)\*\***

**Paste the full DAO UI component here (we’ve already written this).**

**---**

**📦 \*\*2. `SoraDAO.json` (ABI)\*\***

**Ensure your compiled contract ABI (from Hardhat/Remix) is exported like this:**

**```json**

**{**

**"abi": [ ... ]**

**}**

**```**

**Put it inside `/contracts/SoraDAO.json`**

**---**

**📦 \*\*3. `button.tsx` (Reusable UI Button)\*\***

**You can use ShadCN UI or your own:**

**```tsx**

**export function Button({ children, onClick, variant = "default" }) {**

**const styles = variant === "destructive"**

**? "bg-red-500 hover:bg-red-600 text-white px-4 py-2 rounded"**

**: variant === "outline"**

**? "border border-gray-400 px-4 py-2 rounded"**

**: "bg-blue-500 hover:bg-blue-600 text-white px-4 py-2 rounded";**

**return <button onClick={onClick} className={styles}>{children}</button>;**

**}**

**📁 \*\*SORA AI Financial DAO - Backend API Structure\*\***

**```**

**/sora-dao-backend**

**├── /routes**

**│ ├── /proposals**

**│ │ └── create.ts # Create + AI-score a proposal**

**│ └── /ipfs**

**│ └── upload.ts # Upload proposal to IPFS**

**├── /lib**

**│ ├── aiScore.ts # AI scoring function**

**│ └── ipfsClient.ts # Pinata/IPFS integration**

**├── .env # Backend environment vars**

**├── package.json # Backend dependencies**

**├── tsconfig.json # TypeScript config**

**├── README.md**

**```**

**---**

**📦 \*\*1. `/routes/proposals/create.ts`\*\***

**```ts**

**import { scoreProposal } from "../../lib/aiScore";**

**import { uploadToIPFS } from "../../lib/ipfsClient";**

**export async function POST(req) {**

**const { title, description } = await req.json();**

**const score = await scoreProposal(title, description);**

**const ipfsUrl = await uploadToIPFS({ title, description, score });**

**return Response.json({ score, ipfsUrl });**

**}**

**```**

**📦 \*\*2. `lib/aiScore.ts`\*\***

**```ts**

**export async function scoreProposal(title: string, description: string): Promise<number> {**

**const content = `${title}\n\n${description}`;**

**const score = Math.floor(Math.random() \* 100); // Placeholder for AI score logic**

**return score;**

**}**

**```**

**📦 \*\*3. `lib/ipfsClient.ts` (using Pinata)\*\***

**```ts**

**import axios from "axios";**

**const PINATA\_JWT = process.env.PINATA\_JWT || "";**

**export async function uploadToIPFS(data: any): Promise<string> {**

**const res = await axios.post(**

**"https://api.pinata.cloud/pinning/pinJSONToIPFS",**

**data,**

**{ headers: { Authorization: `Bearer ${PINATA\_JWT}` } }**

**);**

**return `https://gateway.pinata.cloud/ipfs/${res.data.IpfsHash}`;**

**}**

**```**

**📦 \*\*4. `.env` Example\*\***

**```**

**PINATA\_JWT=Bearer eyJhbGciOi...**

**```**

**📦 \*\*5. `package**