1. 개요: 기여자 가치를 보장하는 탈중앙화 보상 시스템

MARTHOS는 기여한 만큼 공정한 가치를 인정받고, 성장할 수 있는 **블록체인 기반 보상 시스템**을 구축예정

✔ **모든 기여자의 활동을 자동 기록**
✔ **기여한 만큼 공정한 보상을 분배**
✔ **기여자가 지속적으로 성장할 수 있도록 지원**

---

2. 시스템 아키텍처 개요

핵심 기술 스택

☑ **블록체인 (보안상 공개 불가)** → 스마트 컨트랙트 기반 보상 시스템 구축
☑ **PostgreSQL, MongoDB** → 기여 내역 및 사용자 성장 데이터 저장
☑ **TensorFlow, PyTorch** → 기여 패턴 분석 및 AI 자동 평가 모델
☑ **React, Next.js** → 기여 내역 및 보상 현황 대시보드 구축
☑ **OAuth2, JWT** → 기여 데이터 무결성 및 보안 강화

---

3. DAO 운영 구조 설계

DAO(탈중앙화 자율 조직)를 추가하여 **기여자들이 직접 의사결정에 참여하고 정책을 결정하는 시스템 구축**

📌 **운영 방식:**

☑ **토큰 기반 투표** → ERC-20 or NFT 기반 거버넌스 토큰 발행, 기여도에 따라 투표권 가중치 적용
☑ **제안 시스템** → 기여자들이 정책을 제안하고, 커뮤니티에서 투표
☑ **투표 시스템** → 투표 결과를 **자동 실행하는 스마트 컨트랙트** 개발

---

4. 스마트 컨트랙트 설계 (Solidity)

☑ (1) 기여 내역 블록체인 저장

- 기여자의 활동(코드 개발, 문서 작성, 기부 참여 등)을 **스마트 컨트랙트에 자동 기록**
- 투명한 데이터 저장을 위해 **IPFS(탈중앙 파일 시스템)와 연계**

solidity
예:

```
pragma solidity ^0.8.0;

contract ContributionRecord {
    struct Contribution {
        address contributor;
        string description;
        uint256 timestamp;
        uint256 value;
    }

    Contribution[] public contributions;

    function addContribution(string memory _desc, uint256 _value) public {
```

```
        contributions.push(Contribution(msg.sender, _desc, block.timestamp, _value));
    }

    function getContributions() public view returns (Contribution[] memory) {
        return contributions;
    }
}
```

☑ (2) 투표 시스템 스마트 컨트랙트

✔ 기여자들이 **제안을 올리고 투표 가능**
✔ 특정 투표 수 초과 시 **자동 실행**

solidity
예:

```
pragma solidity ^0.8.0;

contract Governance {
    struct Proposal {
        uint id;
        string description;
        uint voteCount;
        bool executed;
        address proposer;
    }

    mapping(uint =& Proposal) public proposals;
    mapping(address =& mapping(uint =& bool)) public votes;
    uint public proposalCount;

    event ProposalCreated(uint id, string description, address proposer);
    event Voted(uint id, address voter);
    event ProposalExecuted(uint id);

    function createProposal(string memory _description) public {
        proposals[proposalCount] = Proposal(proposalCount, _description, 0, false, msg.sender);
        emit ProposalCreated(proposalCount, _description, msg.sender);
        proposalCount++;
    }

    function vote(uint _id) public {
        require(!votes[msg.sender][_id], "Already voted");
        proposals[_id].voteCount++;
        votes[msg.sender][_id] = true;
        emit Voted(_id, msg.sender);
    }
```

```
    function executeProposal(uint _id) public {
        require(proposals[_id].voteCount & 5, "Not enough votes");
        require(!proposals[_id].executed, "Already executed");

        proposals[_id].executed = true;
        emit ProposalExecuted(_id);
    }
}
```

☑ (3) AI 기반 기여 평가 및 자동 보상 지급

✔ AI가 기여도를 평가하여 **보상 점수 자동 부여**
✔ 스마트 컨트랙트와 연계해 **자동 보상 지급**

python
예:

```
import torch
import torch.nn as nn

# 기여도를 평가하는 간단한 AI 모델
class ContributionScorer(nn.Module):
    def __init__(self):
        super(ContributionScorer, self).__init__()
        self.fc1 = nn.Linear(3, 16)
        self.fc2 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# 예제 입력: [코드 라인 수, 커밋 수, 문서 작성 횟수]
example_input = torch.tensor([[500, 10, 2]], dtype=torch.float32)
model = ContributionScorer()
score = model(example_input)
print("기여 평가 점수:", score.item())
```

✔ **AI 자동 평가 + DAO 투표를 결합하여 보상 결정**
✔ **DAO가 최종 승인하면, 스마트 컨트랙트가 자동 보상 지급**

solidity
예:

```
contract RewardSystem {
    mapping(address =& uint256) public rewards;

    function distributeReward(address _contributor, uint256 _amount) public {
```

```
        rewards[_contributor] += _amount;
    }

    function getRewardBalance(address _contributor) public view returns (uint256) {
        return rewards[_contributor];
    }
}
```

---

5. 프론트엔드 & 백엔드 변경 사항

프론트엔드 (React + Next.js) → DAO 대시보드 추가

- 기여 내역, 제안 리스트, 투표 현황을 실시간 확인 가능
- **투표 UI 추가하여 커뮤니티 참여 가능**

jsx

예:

```
import { useState, useEffect } from "react";

export default function Governance() {
    const [proposals, setProposals] = useState([]);

    useEffect(() =& {
        fetch("/api/proposals")
            .then((res) =& res.json())
            .then((data) =& setProposals(data));
    }, []);

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&거버넌스 제안&/h1&
            {proposals.map((p, index) =& (
                &div key={index} className="border p-2 my-2"&
                    &p&&strong&ID:&/strong& {p.id}&/p&
                    &p&&strong&설명:&/strong& {p.description}&/p&
                    &p&&strong&투표 수:&/strong& {p.voteCount}&/p&
                    &button className="bg-blue-500 text-white px-4 py-2 rounded" onClick={() =& vote(p.id)}&
투표하기&/button&
                &/div&
            ))}
        &/div&
    );
}

async function vote(id) {
    await fetch(`/api/vote/${id}`, { method: "POST" });
    alert("투표 완료!");
}
```

📌 **백엔드 (Node.js + Express.js) → 투표 API 추가**

javascript

예:

```
const express = require("express");
const app = express();

let proposals = [
    { id: 1, description: "기여 보상 증가", voteCount: 3 },
    { id: 2, description: "스테이킹 보상 추가", voteCount: 2 },
];

app.get("/api/proposals", (req, res) =& {
    res.json(proposals);
});

app.post("/api/vote/:id", (req, res) =& {
    const id = parseInt(req.params.id);
    const proposal = proposals.find((p) =& p.id === id);
    if (proposal) {
        proposal.voteCount++;
        res.json({ success: true });
    } else {
        res.status(404).json({ error: "Proposal not found" });
    }
});

app.listen(3000, () =& {
    console.log("Server running on port 3000");
});
```

✔ **기여자는 단순 참여자가 아니라, 시스템을 직접 운영하는 멤버가 됨**

✔ **보상이 투명하게 결정되고, 커뮤니티가 직접 정책을 결정함**

DAO 기술 확장

DAO 거버넌스 강화를 위한 추가 기능

☑ **멀티서명 지갑(Multisig Wallet) 적용**
- 거버넌스 주요 변경 사항은 **멀티서명 승인 후 실행**하도록 보안 강화
- **Gnosis Safe** 같은 **DAO 거버넌스 전용 멀티서명 지갑** 연동 고려

☑ **계층적 투표 시스템(Hierarchical Voting)**
- 기여자의 등급에 따라 **투표 가중치 차등 적용**
- 오래된 기여자는 더 큰 영향력 부여 가능

☑ **자동 펀딩 시스템(DAO Treasury Management)**
- 커뮤니티 기부금 & DAO 자금 관리 시스템 연동
- 특정 투표를 통해 자금 배분 가능

☑ **익명 투표(Private Voting) 지원**
- 블록체인 상에서 **투표 내용이 완전히 공개되는 문제 해결**
- **zk-SNARKs (영지식 증명) 활용 가능**

보상 모델 세분화

☑ **기여 유형별 보상 차등 지급**
- 개발자, 기획자, 홍보 담당자 등 **역할별 기여도 평가 모델** 추가
- 예를 들어, **코드 기여 = AI 코드 품질 분석 + DAO 투표 결합**

☑ **지연 보상(Delayed Vesting) 적용**
- 기여 보상을 일정 기간 후 점진적으로 지급하여 **장기 기여 유도**
- DAO에서 승인된 기여는 **1개월 후 50% 지급 → 3개월 후 100% 지급** 구조

☑ **토큰 스테이킹 보상 추가**
- DAO 토큰을 일정 기간 스테이킹하면 추가 보상 지급
- 예: 6개월 스테이킹 시 **추가 보상 + 거버넌스 권한 증가**

☑ **NFT 보상 시스템 연계**
- 기여도가 높은 멤버에게 **NFT 배지 지급**
- NFT 보유자는 특별한 DAO 기능(예: 상위 투표권) 사용 가능

기술적 확장 가능성

☑ **다중 블록체인 지원 (Multi-Chain DAO) 고려**
- **다른 블록체인 등과의 연결 고려**
- 사용자가 원하는 체인에서 참여할 수 있도록 확장

☑ **IPFS + Filecoin 기반 데이터 저장 보완**
- 기여 내역과 문서 데이터를 **완전 탈중앙화된 저장소(IPFS & Filecoin)에 저장**
- 데이터 변조 방지 및 보존성 강화

☑ **AI 기반 자동 거버넌스 추천 시스템 추가**
- TensorFlow AI 모델이 **기여 패턴 분석 → DAO 투표 추천 시스템** 구현
- 예: "이 사용자는 지난 3개월간 적극적인 기여 → 자동으로 투표 가중치 증가"

✔ DAO의 보안 강화 (멀티서명, 익명 투표)
✔ 기여 보상 모델 세분화 (지연 보상, NFT 보상)
✔ 다중 블록체인 지원 (******** 등 연동)
✔ AI 자동 거버넌스 추천 시스템

1.멀티서명 지갑 (Multisig Wallet) 적용

📌 **목적:**

- DAO의 주요 변경 사항(자금 출금, 정책 변경 등)을 **단일 관리자**가 아니라, 여러 명의 서명(승인)을 받아야 실행되도록 설계
- **해킹이나 내부 부정행위를 방지**하여 DAO의 투명성과 안정성을 강화

📌 **기술 적용:**

☑ **Gnosis Safe** 활용 (EVM 호환 DAO 전용 멀티서명 지갑)

☑ **스마트 컨트랙트 기반 승인 시스템 구축**

☑ **거버넌스 투표 후 일정 비율 이상이 승인해야 실행**

📌 **스마트 컨트랙트 예제 (Solidity)**

solidity

예:

```
pragma solidity ^0.8.0;

contract MultiSigWallet {
    address[] public owners;
    uint public requiredApprovals;

    struct Transaction {
        address to;
        uint amount;
        bool executed;
        uint approvals;
    }

    mapping(uint =& mapping(address =& bool)) public approvals;
    Transaction[] public transactions;

    constructor(address[] memory _owners, uint _requiredApprovals) {
        owners = _owners;
        requiredApprovals = _requiredApprovals;
    }

    function submitTransaction(address _to, uint _amount) public {
        require(isOwner(msg.sender), "Not an owner");
        transactions.push(Transaction(_to, _amount, false, 0));
    }

    function approveTransaction(uint _txIndex) public {
        require(isOwner(msg.sender), "Not an owner");
        require(!approvals[_txIndex][msg.sender], "Already approved");

        approvals[_txIndex][msg.sender] = true;
        transactions[_txIndex].approvals++;

        if (transactions[_txIndex].approvals &= requiredApprovals) {
            executeTransaction(_txIndex);
        }
```

```
    }

    function executeTransaction(uint _txIndex) private {
        require(transactions[_txIndex].approvals &= requiredApprovals, "Not enough approvals");
        require(!transactions[_txIndex].executed, "Already executed");

        transactions[_txIndex].executed = true;
        payable(transactions[_txIndex].to).transfer(transactions[_txIndex].amount);
    }

    function isOwner(address _addr) private view returns (bool) {
        for (uint i = 0; i & owners.length; i++) {
            if (owners[i] == _addr) {
                return true;
            }
        }
        return false;
    }
}
```

✔ DAO에서 **자금 출금이나 정책 실행 시, 여러 명이 서명(승인)해야 함**
✔ **승인된 비율**(예: 3/5) 이상이 되어야 실행 가능
✔ **Gnosis Safe 같은 멀티서명 솔루션과 연동 가능**

2.익명 투표 (Private Voting) 추가

목적

- DAO 투표가 블록체인에 공개되면서 **누가 어떤 선택을 했는지 노출되는 문제** 해결
- **zk-SNARKs(영지식 증명) 기반 익명 투표 시스템** 도입
- **투표 내용은 익명으로 유지하면서, 결과는 검증 가능**

📌 **기술 적용:**
☑ **zk-SNARKs (영지식 증명) 기반 투표**
☑ **Maci (Minimal Anti-Collusion Infrastructure) 연동 가능**
☑ **투표자의 신원을 숨기면서도, 중복 투표를 방지**
📌 **스마트 컨트랙트 예제 (Solidity + zk-SNARKs)**
solidity
예:

```
pragma solidity ^0.8.0;

contract PrivateVoting {
    struct Vote {
        uint proposalId;
        bytes32 encryptedVote;
    }

    mapping(address =& Vote) public votes;
```

```
    mapping(uint =& uint) public results;

    function castVote(uint _proposalId, bytes32 _encryptedVote) public {
        require(votes[msg.sender].proposalId == 0, "Already voted");
        votes[msg.sender] = Vote(_proposalId, _encryptedVote);
    }

    function tallyVotes(uint _proposalId, bytes32[] memory _decryptedVotes) public {
        for (uint i = 0; i & _decryptedVotes.length; i++) {
            if (_decryptedVotes[i] == keccak256(abi.encodePacked("YES"))) {
                results[_proposalId]++;
            }
        }
    }

    function getResults(uint _proposalId) public view returns (uint) {
        return results[_proposalId];
    }
}
```

✔ **투표는 암호화(zk-SNARKs)되어 저장됨 → 투표자의 신원 보호**
✔ **최종 집계만 공개되며, 누가 어떤 선택을 했는지는 숨겨짐**
✔ **Maci 기반 익명 투표 솔루션과 연계 가능**

3.시스템 구현 (프론트엔드 & 백엔드)

프론트엔드 (React + Next.js) → 멀티서명 & 익명 투표 UI 추가
jsx
예:

```
import { useState } from "react";

export default function PrivateVoting() {
    const [vote, setVote] = useState("");

    async function submitVote() {
        const encryptedVote = window.crypto.subtle.digest("SHA-256", new TextEncoder().encode(vote));
        await fetch("/api/vote", {
            method: "POST",
            body: JSON.stringify({ encryptedVote }),
            headers: { "Content-Type": "application/json" },
        });
        alert("투표 완료!");
    }

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&익명 투표&/h1&
```

```
            &select value={vote} onChange={(e) =& setVote(e.target.value)}&
                &option value=""&선택&/option&
                &option value="YES"&찬성&/option&
                &option value="NO"&반대&/option&
            &/select&
            &button className="bg-blue-500 text-white px-4 py-2 rounded" onClick={submitVote}&
                투표 제출
            &/button&
        &/div&
    );
}
```

백엔드 (Node.js + Express.js) → 익명 투표 API 추가

javascript

예:

```
const express = require("express");
const crypto = require("crypto");

const app = express();
app.use(express.json());

let votes = [];

app.post("/api/vote", (req, res) =& {
    const { encryptedVote } = req.body;
    if (!encryptedVote) return res.status(400).json({ error: "투표 데이터 없음" });

    votes.push(encryptedVote);
    res.json({ success: true });
});

app.get("/api/results", (req, res) =& {
    const yesVotes = votes.filter(v =& v === crypto.createHash("sha256").update("YES").digest("hex")).length;
    res.json({ yesVotes });
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

✔ 익명 투표 데이터는 해시(암호화)되어 저장됨
✔ 투표자는 누구나 자신의 투표 기록을 검증 가능하지만, 공개되지 않음
✔ zk-SNARKs 솔루션과 연동하여 보안 강화 가능

DAO 강화: 계층적 투표 시스템 + 자동 펀딩 시스템 추가

☑ **계층적 투표 시스템 (Hierarchical Voting) 추가**
☑ **자동 펀딩 시스템 (DAO Treasury Management) 연동**

---

1. 계층적 투표 시스템 (Hierarchical Voting)

- 기여자의 등급에 따라 **투표 가중치 차등 적용**
- **오래된 기여자, 더 많은 공헌을 한 기여자**에게 더 큰 투표 영향력 부여
- DAO 내에서 **경험과 신뢰를 반영하는 구조 구축**

📌 **기술 적용:**
☑ **기여도 평가 시스템 연동** (AI + 스마트 컨트랙트)
☑ **투표 가중치 적용 (Solidity 기반)**
☑ **오래된 기여자는 더 높은 투표 영향력을 가짐**
📌 **스마트 컨트랙트 예제 (Solidity)**

solidity
예:

```
pragma solidity ^0.8.0;

contract HierarchicalVoting {
    struct Voter {
        uint weight;
        bool voted;
        uint vote;
    }

    struct Proposal {
        uint id;
        string description;
        uint voteCount;
    }

    mapping(address =& Voter) public voters;
    Proposal[] public proposals;

    constructor() {
        // 예시: 관리자(0xAdmin...)가 초기 기여자의 가중치를 설정
        voters[0xAdminAddress] = Voter(5, false, 0);
    }

    function addProposal(string memory _description) public {
        proposals.push(Proposal(proposals.length, _description, 0));
    }

    function vote(uint _proposalId) public {
        require(!voters[msg.sender].voted, "Already voted");
```

```
        require(voters[msg.sender].weight & 0, "No voting power");

        voters[msg.sender].voted = true;
        voters[msg.sender].vote = _proposalId;
        proposals[_proposalId].voteCount += voters[msg.sender].weight;
    }

    function assignVotingWeight(address _voter, uint _weight) public {
        // 관리자만 가중치를 설정할 수 있도록 제한 가능
        voters[_voter].weight = _weight;
    }
}
```

✔ **기여도에 따라 투표 가중치를 다르게 적용**
✔ **오래된 기여자는 자동으로 더 높은 가중치 부여**
✔ **AI 기반으로 가중치 조정 자동화 가능 (예: 기여 내역 분석)**

2.자동 편딩 시스템 (DAO Treasury Management)

- DAO 기부금 & 자금을 관리하고 **투표를 통해 자동 배분**
- 특정 프로젝트에 **투표로 예산을 할당할 수 있도록 설계**
- **재무적 투명성을 보장하고, 탈중앙화된 자금 운용 가능**

기술 적용:
☑ **스마트 컨트랙트 기반 DAO Treasury 관리**
☑ **투표를 통해 자동으로 기부금 배분**
☑ **자금 이동은 다중 서명(Multisig)으로 보호**

스마트 컨트랙트 예제 (Solidity)
solidity
예:

```
pragma solidity ^0.8.0;

contract DAOTreasury {
    address[] public admins;
    uint public requiredApprovals;
    mapping(address =& uint) public balances;
    mapping(uint =& Proposal) public proposals;

    struct Proposal {
        uint id;
        string description;
        uint amount;
        address payable recipient;
        uint voteCount;
```

```
        bool executed;
    }

    uint public proposalCount;

    event ProposalCreated(uint id, string description, uint amount, address recipient);
    event FundsTransferred(uint id, address recipient, uint amount);

    modifier onlyAdmin() {
        require(isAdmin(msg.sender), "Not an admin");
        _;
    }

    constructor(address[] memory _admins, uint _requiredApprovals) {
        admins = _admins;
        requiredApprovals = _requiredApprovals;
    }

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function createProposal(string memory _description, uint _amount, address payable _recipient) public
onlyAdmin {
        require(_amount &= address(this).balance, "Insufficient funds");

        proposals[proposalCount] = Proposal(proposalCount, _description, _amount, _recipient, 0, false);
        emit ProposalCreated(proposalCount, _description, _amount, _recipient);
        proposalCount++;
    }

    function voteOnProposal(uint _id) public {
        require(!proposals[_id].executed, "Already executed");
        proposals[_id].voteCount++;

        if (proposals[_id].voteCount &= requiredApprovals) {
            executeProposal(_id);
        }
    }

    function executeProposal(uint _id) private {
        require(proposals[_id].voteCount &= requiredApprovals, "Not enough votes");
        require(!proposals[_id].executed, "Already executed");

        proposals[_id].executed = true;
        proposals[_id].recipient.transfer(proposals[_id].amount);
        emit FundsTransferred(_id, proposals[_id].recipient, proposals[_id].amount);
    }
```

```
    function isAdmin(address _addr) private view returns (bool) {
        for (uint i = 0; i & admins.length; i++) {
            if (admins[i] == _addr) {
                return true;
            }
        }
        return false;
    }
}
```

✔ **DAO 기부금 관리 + 자동화된 예산 배분 가능**
✔ **기부된 자금은 커뮤니티 투표로 승인된 프로젝트에 배분**
✔ **투표에서 특정 기준을 충족하면 자동으로 실행됨**

3.시스템 구현 (프론트엔드 & 백엔드)

프론트엔드 (React + Next.js) → 투표 및 자금 관리 UI 추가
jsx
예:

```
import { useState } from "react";

export default function Treasury() {
    const [proposal, setProposal] = useState({ description: "", amount: "", recipient: "" });

    async function submitProposal() {
        await fetch("/api/proposals", {
            method: "POST",
            body: JSON.stringify(proposal),
            headers: { "Content-Type": "application/json" },
        });
        alert("제안 제출 완료!");
    }

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&DAO 자금 관리&/h1&
            &input type="text" placeholder="제안 내용" onChange={(e) =& setProposal({ ...proposal, description: e.target.value })} /&
            &input type="number" placeholder="금액" onChange={(e) =& setProposal({ ...proposal, amount: e.target.value })} /&
            &input type="text" placeholder="수혜자 주소" onChange={(e) =& setProposal({ ...proposal, recipient: e.target.value })} /&
            &button className="bg-green-500 text-white px-4 py-2 rounded" onClick={submitProposal}&
                제안 제출
            &/button&
        &/div&
```

```
    );
}
```

📌 **백엔드 (Node.js + Express.js) → 자금 제안 API 추가**

javascript

예:

```
const express = require("express");
const app = express();
app.use(express.json());

let proposals = [];

app.post("/api/proposals", (req, res) =& {
    const { description, amount, recipient } = req.body;
    proposals.push({ id: proposals.length, description, amount, recipient, voteCount: 0 });
    res.json({ success: true });
});

app.post("/api/vote/:id", (req, res) =& {
    const id = parseInt(req.params.id);
    proposals[id].voteCount++;
    res.json({ success: true });
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

✔ **기부금을 모아, 투표를 통해 자동으로 배분 가능**

✔ **제안 제출 및 투표 기능 추가**

✔ **최소 투표 수 도달 시 자동 실행 (Solidity 컨트랙트와 연동 가능)**

DAO 보상 시스템 강화: 기여 유형별 차등 지급 + 지연 보상(Delayed Vesting) 적용

☑ 기여 유형별 보상 차등 지급
☑ 지연 보상(Delayed Vesting) 적용

1.기여 유형별 보상 차등 지급

- **역할별 기여도를 평가하여 공정한 보상을 분배**
- 개발자, 기획자, 홍보 담당자 등 **다양한 기여 유형을 반영하는 보상 모델 구축**
- **코드 품질 평가 + DAO 투표 결합하여 공정성 보장**

📌 **기술 적용:**
☑ **AI 기반 코드 품질 분석 (TensorFlow, PyTorch)**
☑ **DAO 투표 기반 기여 평가 반영 (Solidity)**
☑ **스마트 컨트랙트를 통해 보상 차등 지급**
📌 **스마트 컨트랙트 예제 (Solidity)**
solidity
예:

```
pragma solidity ^0.8.0;

contract ContributionRewards {
    struct Contribution {
        address contributor;
        string role;
        uint score;
        uint reward;
    }

    mapping(address =& Contribution) public contributions;
    uint public developerWeight = 5;
    uint public plannerWeight = 3;
    uint public marketerWeight = 2;

    function submitContribution(string memory _role, uint _score) public {
        uint weight = getRoleWeight(_role);
        uint reward = _score * weight;

        contributions[msg.sender] = Contribution(msg.sender, _role, _score, reward);
    }

    function getRoleWeight(string memory _role) internal view returns (uint) {
        if (keccak256(abi.encodePacked(_role)) == keccak256(abi.encodePacked("Developer"))) {
            return developerWeight;
        } else if (keccak256(abi.encodePacked(_role)) == keccak256(abi.encodePacked("Planner"))) {
```

```
            return plannerWeight;
        } else {
            return marketerWeight;
        }
    }

    function claimReward() public view returns (uint) {
        return contributions[msg.sender].reward;
    }
}
```

✔ 역할별 가중치 적용 (개발자 & 기획자 & 홍보 담당자 순으로 차등 보상)
✔ 기여 점수를 반영하여 보상 계산 후 지급
✔ DAO 투표와 AI 평가 결과를 합산하여 보상 결정 가능

📌 AI 코드 품질 분석 (Python + PyTorch)
python
예:

```
import torch
import torch.nn as nn

class CodeQualityScorer(nn.Module):
    def __init__(self):
        super(CodeQualityScorer, self).__init__()
        self.fc1 = nn.Linear(3, 16)
        self.fc2 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# 예제 입력: [코드 라인 수, 버그 수정 횟수, 코드 리뷰 점수]
example_input = torch.tensor([[500, 3, 8]], dtype=torch.float32)
model = CodeQualityScorer()
score = model(example_input)
print("코드 품질 점수:", score.item())
```

✔ AI가 코드 품질 점수를 평가하여 DAO 보상에 반영 가능
✔ 코드 라인 수, 버그 수정 횟수, 코드 리뷰 점수를 입력하여 점수 생성

---

2. 지연 보상(Delayed Vesting) 적용

- 장기적인 기여를 유도하기 위해 보상을 일정 기간에 걸쳐 점진적으로 지급

- DAO에서 승인된 기여는 1개월 후 50% 지급 → 3개월 후 100% 지급 구조
- 기여자가 단기적인 이익만 추구하지 않도록 설계

기술 적용:

☑ 스마트 컨트랙트 기반 지연 보상 적용
☑ 스테이킹 방식으로 보상을 잠금(Vesting Period 설정)
☑ 특정 기간이 지나면 자동 지급

📌 스마트 컨트랙트 예제 (Solidity)

solidity

예:

```
pragma solidity ^0.8.0;

contract VestingRewards {
    struct Vesting {
        address contributor;
        uint amount;
        uint startTime;
        bool claimed50;
        bool claimed100;
    }

    mapping(address =& Vesting) public vestingSchedules;

    function startVesting(address _contributor, uint _amount) public {
        vestingSchedules[_contributor] = Vesting(_contributor, _amount, block.timestamp, false, false);
    }

    function claimRewards() public {
        Vesting storage vesting = vestingSchedules[msg.sender];
        require(vesting.amount & 0, "No vested tokens");

        uint elapsedTime = block.timestamp - vesting.startTime;

        if (elapsedTime &= 30 days && !vesting.claimed50) {
            payable(msg.sender).transfer(vesting.amount / 2);
            vesting.claimed50 = true;
        }

        if (elapsedTime &= 90 days && !vesting.claimed100) {
            payable(msg.sender).transfer(vesting.amount / 2);
            vesting.claimed100 = true;
        }
    }
}
```

✔ 1개월 후 50% 지급, 3개월 후 100% 지급 구조 적용
✔ DAO에서 승인된 기여자는 자동으로 보상 스케줄 등록됨
✔ 단기 기여만 하고 떠나는 사용자를 방지하는 효과

3.시스템 구현 (프론트엔드 & 백엔드)

프론트엔드 (React + Next.js) → 기여 & 보상 확인 UI 추가

jsx

예:

```
import { useState, useEffect } from "react";

export default function Rewards() {
    const [reward, setReward] = useState(0);

    useEffect(() =& {
        fetch("/api/reward")
            .then((res) =& res.json())
            .then((data) =& setReward(data.amount));
    }, []);

    async function claimReward() {
        await fetch("/api/claim", { method: "POST" });
        alert("보상이 청구되었습니다!");
    }

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&보상 관리&/h1&
            &p&현재 보상 금액: {reward} 토큰&/p&
            &button className="bg-green-500 text-white px-4 py-2 rounded" onClick={claimReward}&
                보상 청구
            &/button&
        &/div&
    );
}
```

📌 백엔드 (Node.js + Express.js) → 보상 API 추가

javascript

예:

```
const express = require("express");
const app = express();
app.use(express.json());

let rewards = {
    "0xUserAddress": { amount: 100, claimed50: false, claimed100: false, startTime: Date.now() }
};

app.get("/api/reward", (req, res) =& {
    res.json(rewards["0xUserAddress"]);
});
```

```
app.post("/api/claim", (req, res) =& {
    let userReward = rewards["0xUserAddress"];
    let elapsedTime = Date.now() - userReward.startTime;

    if (elapsedTime &= 30 * 24 * 60 * 60 * 1000 && !userReward.claimed50) {
        userReward.amount -= 50;
        userReward.claimed50 = true;
    }

    if (elapsedTime &= 90 * 24 * 60 * 60 * 1000 && !userReward.claimed100) {
        userReward.amount -= 50;
        userReward.claimed100 = true;
    }

    res.json({ success: true });
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

✔ **기여 유형별 보상 차등 지급 & 지연 보상 API 적용**
✔ **기여자는 일정 기간 후 자동으로 보상을 받을 수 있음**
✔ **DAO에서 기여 승인이 완료되면 자동 등록됨**

DAO 보상 시스템 확장: 토큰 스테이킹 + NFT 보상 시스템 연계

☑ 토큰 스테이킹 보상 추가
☑ NFT 보상 시스템 연계

1.토큰 스테이킹 보상 추가

목적:

- DAO 토큰을 일정 기간 동안 스테이킹하면 추가 보상을 지급
- 6개월 이상 스테이킹 시 추가 보상 + 거버넌스 권한 증가
- 장기적인 기여를 유도하고, DAO 운영 안정성 확보

기술 적용:

☑ 스마트 컨트랙트 기반 스테이킹 시스템 구축 (Solidity)
☑ 스테이킹 기간에 따라 보상률 증가 (Vesting 적용)
☑ 거버넌스 투표권 증가 시스템 연계
📌 스마트 컨트랙트 예제 (Solidity)

solidity

예:

```
pragma solidity ^0.8.0;

contract StakingRewards {
    struct Stake {
        uint amount;
        uint startTime;
        bool claimed;
    }

    mapping(address =& Stake) public stakes;
    uint public minStakeTime = 180 days; // 최소 6개월 스테이킹
    uint public rewardMultiplier = 20; // 20% 추가 보상

    function stakeTokens() public payable {
        require(msg.value & 0, "Must stake some tokens");
        stakes[msg.sender] = Stake(msg.value, block.timestamp, false);
    }

    function claimRewards() public {
        Stake storage stake = stakes[msg.sender];
        require(block.timestamp &= stake.startTime + minStakeTime, "Staking period not completed");
        require(!stake.claimed, "Already claimed rewards");

        uint reward = stake.amount * rewardMultiplier / 100;
        payable(msg.sender).transfer(reward);

        stake.claimed = true;
    }
```

```
}
```

✔ 6개월 동안 스테이킹하면 20% 추가 보상 지급
✔ 한 번만 보상을 청구할 수 있도록 설계
✔ DAO 투표권 증가 기능과 연동 가능

2.NFT 보상 시스템 연계

- 기여도가 높은 멤버에게 NFT 배지를 지급하여 차별화된 권한 부여
- NFT 보유자는 특별한 DAO 기능(예: 상위 투표권) 사용 가능
- DAO 내에서 NFT를 통한 멤버십 계층 구조 도입 가능

📌 기술 적용:
☑ ERC-721 기반 NFT 배지 발행
☑ NFT 보유자에게 추가적인 DAO 기능 부여
☑ 특정 역할을 수행한 멤버에게 자동 지급
📌 스마트 컨트랙트 예제 (Solidity)
solidity
예:

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";

contract DAOBadgeNFT is ERC721Enumerable {
    uint public nextTokenId;
    address public admin;

    mapping(address =& bool) public hasBadge;

    constructor() ERC721("DAO Badge", "DBADGE") {
        admin = msg.sender;
    }

    function mintBadge(address _recipient) public {
        require(msg.sender == admin, "Only admin can mint");
        require(!hasBadge[_recipient], "User already has a badge");

        _safeMint(_recipient, nextTokenId);
        hasBadge[_recipient] = true;
        nextTokenId++;
    }
}
```

✔ DAO 관리자가 특정 기여자에게 NFT 배지 발급 가능
✔ NFT는 ERC-721 표준을 기반으로 하며, 거래 가능
✔ NFT를 보유한 기여자는 DAO에서 특별한 기능 사용 가능 (예: 상위 투표권 부여)

DAO 거버넌스와 연결 (NFT 보유자에게 추가 투표권 부여)
solidity
예:
```
contract DAOGovernance {
    mapping(address =& uint) public votingPower;
    address public badgeContract;

    constructor(address _badgeContract) {
        badgeContract = _badgeContract;
    }

    function getVotingPower(address _user) public view returns (uint) {
        return IERC721(badgeContract).balanceOf(_user) & 0 ? 2 : 1;
    }
}
```

✔ **NFT를 보유한 사용자는 기본 투표권의 2배 영향력 행사 가능**
✔ **DAO 내에서 기여도가 높은 사람에게 강한 거버넌스 권한 부여 가능**

3.시스템 구현 (프론트엔드 & 백엔드)

프론트엔드 (React + Next.js) → 스테이킹 & NFT 보상 UI 추가
jsx
예:
```
import { useState } from "react";

export default function Staking() {
    const [staked, setStaked] = useState(false);

    async function stakeTokens() {
        await fetch("/api/stake", { method: "POST" });
        setStaked(true);
        alert("토큰 스테이킹 완료!");
    }

    async function claimRewards() {
        await fetch("/api/claim", { method: "POST" });
        alert("보상이 청구되었습니다!");
    }

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&스테이킹 & NFT 보상&/h1&
            {!staked ? (
                &button className="bg-blue-500 text-white px-4 py-2 rounded" onClick={stakeTokens}&
                    토큰 스테이킹
```

```
                &/button&
            ) : (
                &button className="bg-green-500 text-white px-4 py-2 rounded" onClick={claimRewards}&
                    보상 청구
                &/button&
            )}
        &/div&
    );
}
```

백엔드 (Node.js + Express.js) → 스테이킹 & 보상 API 추가

javascript

예:

```
const express = require("express");
const app = express();
app.use(express.json());

let stakes = {
    "0xUserAddress": { amount: 100, startTime: Date.now(), claimed: false }
};

app.post("/api/stake", (req, res) =& {
    stakes["0xUserAddress"] = { amount: 100, startTime: Date.now(), claimed: false };
    res.json({ success: true });
});

app.post("/api/claim", (req, res) =& {
    let userStake = stakes["0xUserAddress"];
    let elapsedTime = Date.now() - userStake.startTime;

    if (elapsedTime &= 180 * 24 * 60 * 60 * 1000 && !userStake.claimed) {
        userStake.claimed = true;
        res.json({ success: true, reward: userStake.amount * 0.2 });
    } else {
        res.status(400).json({ error: "Staking period not completed" });
    }
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

✔ **사용자가 토큰을 스테이킹하면 자동으로 6개월 후 보상 지급 가능**
✔ **DAO에서 보유 NFT에 따라 추가 거버넌스 권한 부여 가능**

다중 블록체인 지원 (Multi-Chain DAO) 구축

☑ 다중 블록체인 지원 (Multi-Chain DAO)
☑ 사용자가 원하는 체인에서 DAO 참여 가능

◇ 1.다중 블록체인 지원 개념

목적:

- DAO를 ********** 등 다양한 체인과 연동
- 사용자가 자신이 선호하는 블록체인에서 DAO에 참여 가능
- Cross-Chain Messaging (다중 블록체인 메시징) 기술 활용하여 네트워크 간 데이터 공유

기술 적용:

☑ LayerZero, Axelar, Wormhole 같은 Cross-Chain 솔루션 활용
☑ ERC-20 토큰 & NFT가 여러 블록체인에서 사용 가능하도록 브릿지 연동
☑ 스마트 컨트랙트 + 오라클(Chainlink 등) 활용하여 다중 체인 데이터 공유

◇ 2.다중 블록체인 스마트 컨트랙트 설계 (Solidity)

📌 다중 체인에서 DAO 거버넌스 투표를 동기화하는 예제
solidity
예:

```
pragma solidity ^0.8.0;

interface IBridge {
    function sendMessage(address _destination, uint _proposalId, uint _votes) external;
}

contract MultiChainDAO {
    struct Proposal {
        uint id;
        string description;
        uint voteCount;
    }

    mapping(uint =& Proposal) public proposals;
    IBridge public bridgeContract;

    constructor(address _bridgeContract) {
        bridgeContract = IBridge(_bridgeContract);
    }

    function createProposal(string memory _description) public {
        uint proposalId = uint(keccak256(abi.encodePacked(_description, block.timestamp)));
        proposals[proposalId] = Proposal(proposalId, _description, 0);
    }
```

```
    function voteOnProposal(uint _proposalId) public {
        proposals[_proposalId].voteCount++;
        bridgeContract.sendMessage(address(this), _proposalId, proposals[_proposalId].voteCount);
    }
}
```

✔ DAO 투표 데이터를 여러 체인과 동기화
✔ Cross-Chain Bridge를 활용하여 다른 네트워크에서도 동일한 투표 결과 반영
✔ Ethereum에서 투표하면, Polygon에서도 동일한 투표 결과 적용 가능

◇ 3.Cross-Chain 메시징 솔루션 연동

📌 LayerZero를 활용한 다중 체인 메시징 구현

```
solidity
복사편집
pragma solidity ^0.8.0;

import "@layerzero/contracts/interfaces/ILayerZeroEndpoint.sol";

contract CrossChainDAO {
    ILayerZeroEndpoint public endpoint;

    constructor(address _endpoint) {
        endpoint = ILayerZeroEndpoint(_endpoint);
    }

    function sendVoteResult(uint16 _dstChainId, bytes memory _payload) public {
        endpoint.send{value: msg.value}(
            _dstChainId,
            abi.encodePacked(msg.sender),
            _payload,
            payable(msg.sender),
            address(0x0),
            bytes("")
        );
    }
}
```

📌 설명:
✔ LayerZero를 활용해 다른 블록체인에 DAO 투표 결과 전달
✔ Ethereum → Polygon, Solana, Arbitrum 등 여러 체인과 연결 가능
✔ 사용자는 원하는 체인에서 DAO에 참여 가능

◇ 4.다중 체인 토큰 브릿지 (ERC-20 & NFT)

📌 ERC-20 기반 DAO 토큰을 여러 블록체인에서 사용할 수 있도록 브릿지 연동

solidity
복사편집
```
pragma solidity ^0.8.0;

contract CrossChainToken {
    mapping(address =& uint) public balances;
    mapping(uint16 =& address) public trustedContracts;

    function deposit(uint _amount, uint16 _dstChainId) public {
        require(balances[msg.sender] &= _amount, "Insufficient balance");
        balances[msg.sender] -= _amount;
        sendCrossChainMessage(_dstChainId, _amount);
    }

    function sendCrossChainMessage(uint16 _dstChainId, uint _amount) private {
        // Cross-Chain Bridge 솔루션을 사용하여 토큰 이동
    }

    function claimTokens(address _receiver, uint _amount) public {
        balances[_receiver] += _amount;
    }
}
```

📌 설명:

✔ Polygon에서 DAO 토큰을 보유하면, Ethereum에서도 동일한 토큰 사용 가능

✔ Cross-Chain 브릿지 솔루션을 활용하여 ERC-20 토큰 이동 지원

✔ NFT도 동일한 방식으로 여러 블록체인에서 사용 가능

◇ 5.프론트엔드 & 백엔드 구현

📌 프론트엔드 (React + Next.js) → 다중 체인 지원 UI 추가

jsx
복사편집
```
import { useState } from "react";

export default function MultiChainDAO() {
    const [selectedChain, setSelectedChain] = useState("Ethereum");

    async function voteOnProposal() {
        await fetch(`/api/vote?chain=${selectedChain}`, { method: "POST" });
        alert(`${selectedChain} 체인에서 투표 완료!`);
    }

    return (
        &div className="p-4"&
```

```
            &h1 className="text-xl font-bold"&다중 블록체인 DAO&/h1&
            &select value={selectedChain} onChange={(e) =& setSelectedChain(e.target.value)}&
                &option value="Ethereum"&Ethereum&/option&
                &option value="Polygon"&Polygon&/option&
                &option value="Solana"&Solana&/option&
            &/select&
            &button className="bg-blue-500 text-white px-4 py-2 rounded" onClick={voteOnProposal}&
                투표하기
            &/button&
        &/div&
    );
}
```

📌 **설명:**
✔ **사용자가 원하는 블록체인을 선택하고 투표 가능**
✔ **프론트엔드에서 다중 체인 지원을 쉽게 확장 가능**
📌 **백엔드 (Node.js + Express.js) → 체인별 API 연동**

javascript
복사편집

```
const express = require("express");
const app = express();
app.use(express.json());

app.post("/api/vote", (req, res) =& {
    const { chain } = req.query;
    console.log(`투표가 ${chain} 체인에서 실행되었습니다.`);
    res.json({ success: true });
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

📌 **설명:**
✔ **백엔드에서 사용자의 블록체인 선택에 따라 투표 실행**
✔ **DAO가 다중 체인을 지원하도록 확장 가능**

📌 **사용자가 원하는 체인에서 DAO에 참여 가능 (Ethereum, Polygon, Solana 등)**
📌 **투표 및 토큰 거래를 여러 블록체인에서 동기화**
📌 **Cross-Chain 메시징 기술로 DAO의 확장성 증가**

DAO 데이터 저장 강화: IPFS + Filecoin 기반 완전 탈중앙화 저장소 구축

☑ **IPFS + Filecoin 기반 데이터 저장**
☑ **기여 내역, 문서 데이터를 완전 탈중앙화하여 보존성 강화**
☑ **데이터 변조 방지 및 보안성 극대화**

◇ 1.IPFS + Filecoin을 사용하는 이유

📌 **문제점:**
- 기존 데이터 저장 방식(예: 중앙화된 서버, 블록체인 자체 저장)은 **변조 위험**과 **고비용**이 발생
- 블록체인에 모든 데이터를 직접 저장하면 **가스 비용**이 급증

📌 **해결책:**
☑ **IPFS(InterPlanetary File System) → 데이터의 고유 해시값을 생성하여 분산 저장**
☑ **Filecoin → IPFS에 저장된 데이터를 인센티브 기반으로 장기 보관 보장**
☑ **블록체인에는 데이터 해시값만 저장 → 가스 비용 절감 + 변조 방지**
📌 **실제 적용 방식:**
기여 내역, 문서를 IPFS에 저장 → 해시값 반환
해시값을 Filecoin에 백업하여 장기 보존
해시값을 블록체인 스마트 컨트랙트에 기록

◇ 2.스마트 컨트랙트 설계 (Solidity)

📌 **IPFS 기반 DAO 문서 저장 및 조회 예제**
solidity
복사편집
```
pragma solidity ^0.8.0;

contract DAOStorage {
    struct Document {
        string ipfsHash;
        address uploader;
        uint timestamp;
    }

    mapping(string => Document) public documents;

    event DocumentUploaded(string ipfsHash, address uploader, uint timestamp);

    function uploadDocument(string memory _ipfsHash) public {
        require(bytes(documents[_ipfsHash].ipfsHash).length == 0, "Document already exists");
        documents[_ipfsHash] = Document(_ipfsHash, msg.sender, block.timestamp);
        emit DocumentUploaded(_ipfsHash, msg.sender, block.timestamp);
    }

    function getDocument(string memory _ipfsHash) public view returns (Document memory) {
```

```
        return documents[_ipfsHash];
    }
}
```

📌 **설명:**
✔ **DAO 문서를 IPFS에 저장하고, 해시값만 블록체인에 저장**
✔ **누구나 블록체인을 통해 해당 문서의 무결성 검증 가능**
✔ **Filecoin을 활용해 장기 보존 가능**

◇ 3.Filecoin과 연동하여 데이터 장기 저장

📌 **Filecoin에 DAO 데이터를 저장하는 방식**

- IPFS에 DAO 데이터를 저장하면, **일정 시간이 지나면 사라질 가능성**이 있음
- Filecoin을 활용하면 **경제적 인센티브 기반으로 장기 저장 가능**

📌 **Filecoin 연동 예제 (Powergate API 사용)**

python
복사편집

```
import requests

# Powergate API 엔드포인트 설정 (Filecoin 네트워크 연동)
POWERGATE_HOST = "http://localhost:6002"

def upload_to_filecoin(ipfs_cid):
    url = f"{POWERGATE_HOST}/api/v0/pow/filecoin/storage"
    payload = {"cid": ipfs_cid}
    headers = {"Content-Type": "application/json"}

    response = requests.post(url, json=payload, headers=headers)
    return response.json()

ipfs_hash = "QmXkK5F..."
filecoin_response = upload_to_filecoin(ipfs_hash)
print("Filecoin 저장 결과:", filecoin_response)
```

📌 **설명:**
✔ **IPFS 해시값을 Filecoin에 저장하여 장기 보관**
✔ **Filecoin 저장 성공 시, 블록체인에서 검증 가능**
✔ **DAO 문서를 영구적으로 보존하여 변조 방지**

◇ 4.프론트엔드 & 백엔드 구현

📌 **프론트엔드 (React + Next.js) → DAO 문서 업로드 UI 추가**

jsx
복사편집

```
import { useState } from "react";
```

```
export default function DocumentUpload() {
    const [file, setFile] = useState(null);
    const [ipfsHash, setIpfsHash] = useState("");

    async function uploadToIPFS() {
        const formData = new FormData();
        formData.append("file", file);

        const res = await fetch("https://ipfs.infura.io:5001/api/v0/add", {
            method: "POST",
            body: formData,
        });

        const data = await res.json();
        setIpfsHash(data.Hash);
    }

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&DAO 문서 업로드&/h1&
            &input type="file" onChange={(e) =& setFile(e.target.files[0])} /&
            &button className="bg-blue-500 text-white px-4 py-2 rounded" onClick={uploadToIPFS}&
                IPFS에 업로드
            &/button&
            {ipfsHash && &p&IPFS 해시: {ipfsHash}&/p&}
        &/div&
    );
}
```

📌 **설명:**
✔ **사용자가 DAO 문서를 업로드하면 IPFS에 저장됨**
✔ **저장된 IPFS 해시값을 블록체인에 기록 가능**
📌 **백엔드 (Node.js + Express.js) → IPFS & Filecoin 저장 API**

javascript
복사편집

```
const express = require("express");
const FormData = require("form-data");
const fetch = require("node-fetch");

const app = express();
app.use(express.json());

async function uploadToIPFS(file) {
    const formData = new FormData();
    formData.append("file", file);

    const response = await fetch("https://ipfs.infura.io:5001/api/v0/add", {
        method: "POST",
```

```
        body: formData,
    });

    return response.json();
}

app.post("/api/upload", async (req, res) =& {
    const ipfsResponse = await uploadToIPFS(req.body.file);
    res.json({ ipfsHash: ipfsResponse.Hash });
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

📌 **설명:**
✔ **IPFS + Filecoin API를 활용해 DAO 문서를 업로드**
✔ **IPFS 저장 후, 해시값을 블록체인에 기록 가능**

📌 **IPFS + Filecoin을 사용하여 DAO 문서를 영구 보존 가능**
📌 **블록체인에 해시값만 저장하여 비용 절감 + 변조 방지**
📌 **DAO 데이터가 중앙 서버 없이 안전하게 보호됨**

AI 기반 자동 거버넌스 추천 시스템 구축

☑ TensorFlow 기반 기여 패턴 분석 모델 개발
☑ DAO 내에서 기여도가 높은 멤버의 투표 가중치 자동 조정
☑ 스마트 컨트랙트와 연동하여 자동 반영

◇ 1.AI 기반 자동 거버넌스 추천 시스템 개념

📌 목적:
- DAO 참여자의 기여 패턴을 분석하여 **투표 가중치를 자동 조정**
- **지난 3개월간의 기여 데이터를 분석하여 투표 가중치 증가 여부 판단**
- **장기적으로 기여한 사람들에게 더 많은 영향력을 부여**

📌 기술 적용:
☑ TensorFlow/PyTorch를 활용한 기여 패턴 분석 모델 구축
☑ 스마트 컨트랙트와 연동하여 투표 가중치 자동 조정
☑ DAO의 의사결정을 자동화하여 공정성 강화

◇ 2.AI 모델 설계 (Python + TensorFlow)

📌 **기여도를 분석하여 투표 가중치를 예측하는 머신러닝 모델**

python
복사편집

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

# 기여 패턴을 학습하는 신경망 모델
model = keras.Sequential([
    keras.layers.Dense(16, activation="relu", input_shape=(3,)),  # 기여도 입력 (커밋 수, 제안 개수, 문서 작성 수)
    keras.layers.Dense(8, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid")  # 투표 가중치 증가 여부 (0~1 확률)
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

# 예제 데이터 (커밋 수, 제안 개수, 문서 작성 수 → 가중치 증가 여부)
train_data = np.array([
    [50, 5, 3], [30, 3, 1], [70, 10, 5], [10, 1, 0], [90, 12, 8]
])
train_labels = np.array([1, 0, 1, 0, 1])  # 1 = 가중치 증가, 0 = 유지

# 모델 학습
model.fit(train_data, train_labels, epochs=50, verbose=1)

# 새로운 기여도를 입력하면 투표 가중치 증가 여부 예측
```

```
test_data = np.array([[60, 6, 4]])  # 새로운 사용자의 기여 패턴
prediction = model.predict(test_data)
print("투표 가중치 증가 확률:", prediction)
```

📌 **설명:**
✔ **기여도(커밋 수, 제안 개수, 문서 작성 수) 데이터를 학습하여 투표 가중치 증가 여부를 예측**
✔ **TensorFlow 모델이 특정 사용자가 충분한 기여를 했는지 분석하여 투표 가중치를 자동 조정**
✔ **DAO 거버넌스 시스템에 연동 가능**

◇ 3.스마트 컨트랙트 연동 (Solidity)

📌 **AI 모델이 추천한 투표 가중치를 스마트 컨트랙트에 반영**

```
solidity
복사편집
pragma solidity ^0.8.0;

contract DAOGovernance {
    struct Member {
        address memberAddress;
        uint commitCount;
        uint proposalCount;
        uint docContribution;
        uint votingPower;
    }

    mapping(address =& Member) public members;

    event VotingPowerUpdated(address indexed member, uint newPower);

    function updateVotingPower(address _member, uint _newPower) public {
        require(_newPower &= 1, "Voting power must be at least 1");
        members[_member].votingPower = _newPower;
        emit VotingPowerUpdated(_member, _newPower);
    }

    function getVotingPower(address _member) public view returns (uint) {
        return members[_member].votingPower;
    }
}
```

📌 **설명:**
✔ **AI가 분석한 투표 가중치를 스마트 컨트랙트에서 자동 반영**
✔ **기여도가 높은 사용자는 투표 영향력이 증가**
✔ **기본 투표권(1) → AI 모델이 추천하면 증가 가능**

◇ 4.프론트엔드 & 백엔드 구현

📌 **프론트엔드 (React + Next.js) → AI 기반 투표 가중치 반영 UI**

jsx

복사편집

```
import { useState, useEffect } from "react";

export default function VotingPower() {
    const [votingPower, setVotingPower] = useState(1);

    useEffect(() =& {
        fetch("/api/voting-power")
            .then((res) =& res.json())
            .then((data) =& setVotingPower(data.power));
    }, []);

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&투표 가중치&/h1&
            &p&현재 투표 가중치: {votingPower}&/p&
        &/div&
    );
}
```

📌 **설명:**

✔ **사용자의 투표 가중치를 실시간으로 확인 가능**

✔ **AI 모델이 추천한 값을 자동으로 반영**

📌 **백엔드 (Node.js + Express.js) → AI 분석 & 스마트 컨트랙트 연동**

javascript

복사편집

```
const express = require("express");
const Web3 = require("web3");
const { predictVotingPower } = require("./aiModel");

const app = express();
const web3 = new Web3("https://polygon-rpc.com");

const daoContract = new web3.eth.Contract(DAO_ABI, DAO_ADDRESS);

app.get("/api/voting-power", async (req, res) =& {
    const userAddress = "0xUserAddress";
    const contributionData = await getUserContributionData(userAddress);
    const predictedPower = await predictVotingPower(contributionData);

    await daoContract.methods.updateVotingPower(userAddress, predictedPower).send({ from: ADMIN_ADDRESS
});

    res.json({ power: predictedPower });
});
```

```
async function getUserContributionData(userAddress) {
    // 예제: 블록체인에서 기여 데이터를 가져오는 함수
    return [50, 5, 3]; // (커밋 수, 제안 개수, 문서 작성 수)
}

app.listen(3000, () =& console.log("Server running on port 3000"));
```

📌 **설명:**
✔ **사용자의 기여 데이터를 가져와 AI 모델이 투표 가중치 추천**
✔ **스마트 컨트랙트에 반영하여 DAO 내 자동 투표 시스템 구축 가능**

📌 **TensorFlow 모델이 기여 패턴 분석 → DAO 투표 추천 시스템 구현**
📌 **스마트 컨트랙트와 연동하여 자동으로 투표 가중치 증가**
📌 **DAO가 공정하고 자동화된 거버넌스를 유지할 수 있도록 지원**

추가사항

DAO 온체인 평판 시스템 추가

이제 **온체인 평판 시스템을 도입하여 기여자의 신뢰도를 평가하고, DAO 내에서 장기적인 영향력을 확보하도록** 만들 차례야!

☑ **온체인 평판 시스템 구축 (On-Chain Reputation System)**
☑ **기여 내역을 분석하여 신뢰도를 정량적으로 평가**
☑ **스마트 컨트랙트와 연동하여 평판 점수 반영 & DAO 내 권한 증가**

◇ 1.온체인 평판 시스템 개념

📌 **목적:**

- **DAO 기여도를 장기적으로 평가하여 신뢰도를 관리**
- **기여자가 꾸준히 활동하면 더 높은 권한 & 신뢰 점수 부여**
- **악의적인 행동(예: 부정 투표, 사기 등)을 방지하는 견제 시스템 추가**

📌 **기술 적용:**

☑ **블록체인에 온체인 평판 점수 저장 (Solidity 스마트 컨트랙트)**
☑ **AI 기반 평판 점수 조정 (TensorFlow 모델 연동 가능)**
☑ **DAO 내 거버넌스 기능과 연계 (예: 신뢰 점수가 높은 사용자는 더 많은 권한 부여)**

◇ 2.스마트 컨트랙트 설계 (Solidity)

📌 **기여 내역을 기반으로 평판 점수를 관리하는 스마트 컨트랙트**

solidity
복사편집

```
pragma solidity ^0.8.0;

contract ReputationSystem {
    struct Member {
        address memberAddress;
        uint reputationScore;
        uint contributions;
        uint votesParticipated;
    }

    mapping(address =& Member) public members;

    event ReputationUpdated(address indexed member, uint newScore);

    function updateReputation(address _member, uint _contributions, uint _votes) public {
        uint newScore = _contributions * 10 + _votes * 5; // 기여 & 투표 참여도 반영
        members[_member] = Member(_member, newScore, _contributions, _votes);
        emit ReputationUpdated(_member, newScore);
    }

    function getReputationScore(address _member) public view returns (uint) {
```

```
        return members[_member].reputationScore;
    }
}
```

📌 **설명:**
✔ **기여 내역 & 투표 참여도를 반영하여 평판 점수 계산**
✔ **기여도가 높을수록 더 높은 평판 점수를 부여**
✔ **스마트 컨트랙트에서 평판 점수를 조회 가능**

◇ 3.AI 기반 평판 점수 조정 (Python + TensorFlow)

📌 **기여자의 활동 데이터를 분석하여 신뢰 점수 자동 조정**

python
복사편집

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

# 평판 점수를 예측하는 모델
model = keras.Sequential([
    keras.layers.Dense(16, activation="relu", input_shape=(3,)),  # 기여 데이터 입력
    keras.layers.Dense(8, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid")  # 신뢰 점수 (0~1 확률)
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

# 예제 데이터 (기여 내역, 투표 참여도, 기부 금액)
train_data = np.array([
    [50, 5, 100], [30, 3, 20], [70, 10, 200], [10, 1, 5], [90, 12, 300]
])
train_labels = np.array([1, 0, 1, 0, 1])  # 1 = 높은 평판, 0 = 낮은 평판

# 모델 학습
model.fit(train_data, train_labels, epochs=50, verbose=1)

# 새로운 사용자의 기여 데이터를 입력하면 신뢰 점수 예측
test_data = np.array([[60, 6, 150]])  # 새로운 사용자의 기여 패턴
prediction = model.predict(test_data)
print("신뢰 점수 증가 확률:", prediction)
```

📌 **설명:**
✔ **기여 내역, 투표 참여도, 기부 금액을 분석하여 신뢰 점수 조정**
✔ **AI 모델이 사용자 활동을 평가하고 자동으로 평판 점수 조정 가능**
✔ **스마트 컨트랙트와 연동하여 자동 반영 가능**

◇ 4. DAO 거버넌스 시스템과 연동

📌 **평판 점수를 활용하여 DAO 내에서 추가 권한 부여**

solidity
복사편집
```
contract DAOGovernance {
    mapping(address =& uint) public reputationScore;
    mapping(address =& uint) public votingPower;

    function updateVotingPower(address _member, uint _newReputation) public {
        uint newPower = _newReputation / 10; // 평판 점수에 비례하여 투표 가중치 증가
        votingPower[_member] = newPower;
    }

    function getVotingPower(address _member) public view returns (uint) {
        return votingPower[_member];
    }
}
```

📌 **설명:**

✔ **평판 점수가 높은 사용자는 더 높은 투표 가중치 부여**

✔ **DAO 내에서 신뢰 점수를 기반으로 의사결정 권한 차등 적용 가능**

◇ 5.프론트엔드 & 백엔드 구현

📌 **프론트엔드 (React + Next.js) → 평판 점수 확인 UI**

jsx
복사편집
```
import { useState, useEffect } from "react";

export default function Reputation() {
    const [reputation, setReputation] = useState(0);

    useEffect(() =& {
        fetch("/api/reputation")
            .then((res) =& res.json())
            .then((data) =& setReputation(data.score));
    }, []);

    return (
        &div className="p-4"&
            &h1 className="text-xl font-bold"&온체인 평판 점수&/h1&
            &p&현재 평판 점수: {reputation}&/p&
        &/div&
    );
}
```

📌 **설명:**
✔ **사용자가 자신의 평판 점수를 조회 가능**
✔ **DAO 거버넌스 시스템과 연동하여 실시간 반영 가능**
📌 **백엔드 (Node.js + Express.js) → AI 분석 & 스마트 컨트랙트 연동**

javascript
복사편집

```
const express = require("express");
const Web3 = require("web3");
const { predictReputationScore } = require("./aiModel");

const app = express();
const web3 = new Web3("https://polygon-rpc.com");

const daoContract = new web3.eth.Contract(DAO_ABI, DAO_ADDRESS);

app.get("/api/reputation", async (req, res) => {
    const userAddress = "0xUserAddress";
    const contributionData = await getUserContributionData(userAddress);
    const predictedScore = await predictReputationScore(contributionData);

    await daoContract.methods.updateVotingPower(userAddress, predictedScore).send({ from: ADMIN_ADDRESS
});

    res.json({ score: predictedScore });
});

async function getUserContributionData(userAddress) {
    return [50, 5, 100]; // (기여 내역, 투표 참여도, 기부 금액)
}

app.listen(3000, () => console.log("Server running on port 3000"));
```

📌 **설명:**
✔ **사용자의 기여 내역을 분석하여 AI 모델이 신뢰 점수를 예측**
✔ **DAO 스마트 컨트랙트에 반영하여 온체인 평판 점수 업데이트 가능**

📌 **온체인에서 기여 내역을 기록하고 신뢰 점수 자동 반영 가능**
📌 **평판 점수가 높은 사용자는 더 높은 투표 가중치 & 거버넌스 권한 부여**
📌 **DAO가 장기적으로 신뢰 기반으로 운영될 수 있도록 시스템 강화**

사용자 맞춤형 DAO 대시보드 추가

☑ 사용자의 활동 데이터를 시각화하여 한눈에 확인 가능
☑ 기여 내역, 평판 점수, 투표 가중치, 보상 내역을 통합 관리
☑ DAO의 운영을 더 투명하게 하여 사용자 경험 개선

◇ 1.DAO 대시보드 개념

📌 목적:
- 각 사용자의 DAO 활동 데이터를 실시간으로 제공
- 기여 내역, 평판 점수, 투표 가중치, 보상 내역을 한 곳에서 관리
- 사용자 맞춤형 인터페이스로 DAO의 투명성 & 접근성 개선

📌 기술 적용:
☑ React + Next.js 기반 인터페이스 구축
☑ Node.js + Express.js로 DAO 데이터 API 제공
☑ Web3.js를 활용하여 블록체인에서 데이터 실시간 조회

◇ 2.스마트 컨트랙트 연동 (Solidity)

📌 DAO 기여 내역 & 보상 내역을 조회하는 스마트 컨트랙트

```
solidity
복사편집
pragma solidity ^0.8.0;

contract DAODashboard {
    struct User {
        address userAddress;
        uint reputationScore;
        uint votingPower;
        uint rewards;
    }

    mapping(address =& User) public users;

    function getUserData(address _user) public view returns (uint, uint, uint) {
        return (users[_user].reputationScore, users[_user].votingPower, users[_user].rewards);
    }
}
```

📌 설명:
✔ 사용자의 평판 점수, 투표 가중치, 보상 내역을 조회 가능
✔ DAO 대시보드에서 데이터를 실시간으로 가져올 수 있도록 설계

◇ 3.프론트엔드: DAO 대시보드 UI 구축 (React + Next.js)

📌 사용자가 DAO 활동을 쉽게 확인할 수 있는 맞춤형 인터페이스

jsx

복사편집

```
import { useState, useEffect } from "react";

export default function DAODashboard() {
    const [userData, setUserData] = useState({ reputation: 0, votingPower: 0, rewards: 0 });

    useEffect(() =& {
        fetch("/api/user-data")
            .then((res) =& res.json())
            .then((data) =& setUserData(data));
    }, []);

    return (
        &div className="p-6"&
            &h1 className="text-2xl font-bold"&🚀 내 DAO 활동 대시보드&/h1&
            &div className="grid grid-cols-3 gap-4 mt-4"&
                &div className="border p-4"&
                    &h2 className="text-xl font-semibold"&평판 점수&/h2&
                    &p className="text-3xl"&{userData.reputation}&/p&
                &/div&
                &div className="border p-4"&
                    &h2 className="text-xl font-semibold"&투표 가중치&/h2&
                    &p className="text-3xl"&{userData.votingPower}&/p&
                &/div&
                &div className="border p-4"&
                    &h2 className="text-xl font-semibold"&보상 (토큰)&/h2&
                    &p className="text-3xl"&{userData.rewards}&/p&
                &/div&
            &/div&
        &/div&
    );
}
```

📌 설명:

✔ 사용자의 DAO 기여 내역을 한눈에 볼 수 있도록 시각화

✔ 평판 점수, 투표 가중치, 보상 내역을 실시간 업데이트

✔ Tailwind CSS를 활용하여 깔끔한 UI 제공

◇ 4.백엔드: 사용자 DAO 데이터 API (Node.js + Express.js)

📌 DAO 스마트 컨트랙트에서 사용자 데이터를 가져와 API로 제공

javascript

복사편집

```
const express = require("express");
```

```
const Web3 = require("web3");

const app = express();
const web3 = new Web3("https://polygon-rpc.com");

const daoContract = new web3.eth.Contract(DAO_ABI, DAO_ADDRESS);

app.get("/api/user-data", async (req, res) =& {
    const userAddress = "0xUserAddress";
    const userData = await daoContract.methods.getUserData(userAddress).call();

    res.json({
        reputation: userData[0],
        votingPower: userData[1],
        rewards: userData[2]
    });
});

app.listen(3000, () =& console.log("Server running on port 3000"));
```

📌 **설명:**
✔ **DAO 스마트 컨트랙트에서 사용자의 기여 데이터를 가져와 API로 제공**
✔ **React 대시보드와 연동하여 실시간 업데이트 가능**

◇ 5.추가 기능 (추후 확장 가능)

📌 **사용자 맞춤형 기능 추가 가능**
☑ **기여 내역 상세 조회 (커밋 수, 투표 참여 횟수, 기부 금액 등)**
☑ **DAO 거버넌스 제안 & 투표 내역 조회**
☑ **보상 이력 그래프 표시 (과거 보상 내역 추적)**
☑ **투표 내역 분석 (어떤 제안에 가장 많이 참여했는지 분석)**

📌 **사용자가 자신의 DAO 활동을 실시간으로 조회 가능**
📌 **투명한 DAO 운영을 통해 신뢰성 & 사용자 경험 개선**
📌 **투표 가중치, 평판 점수, 보상 내역을 한 곳에서 확인 가능**