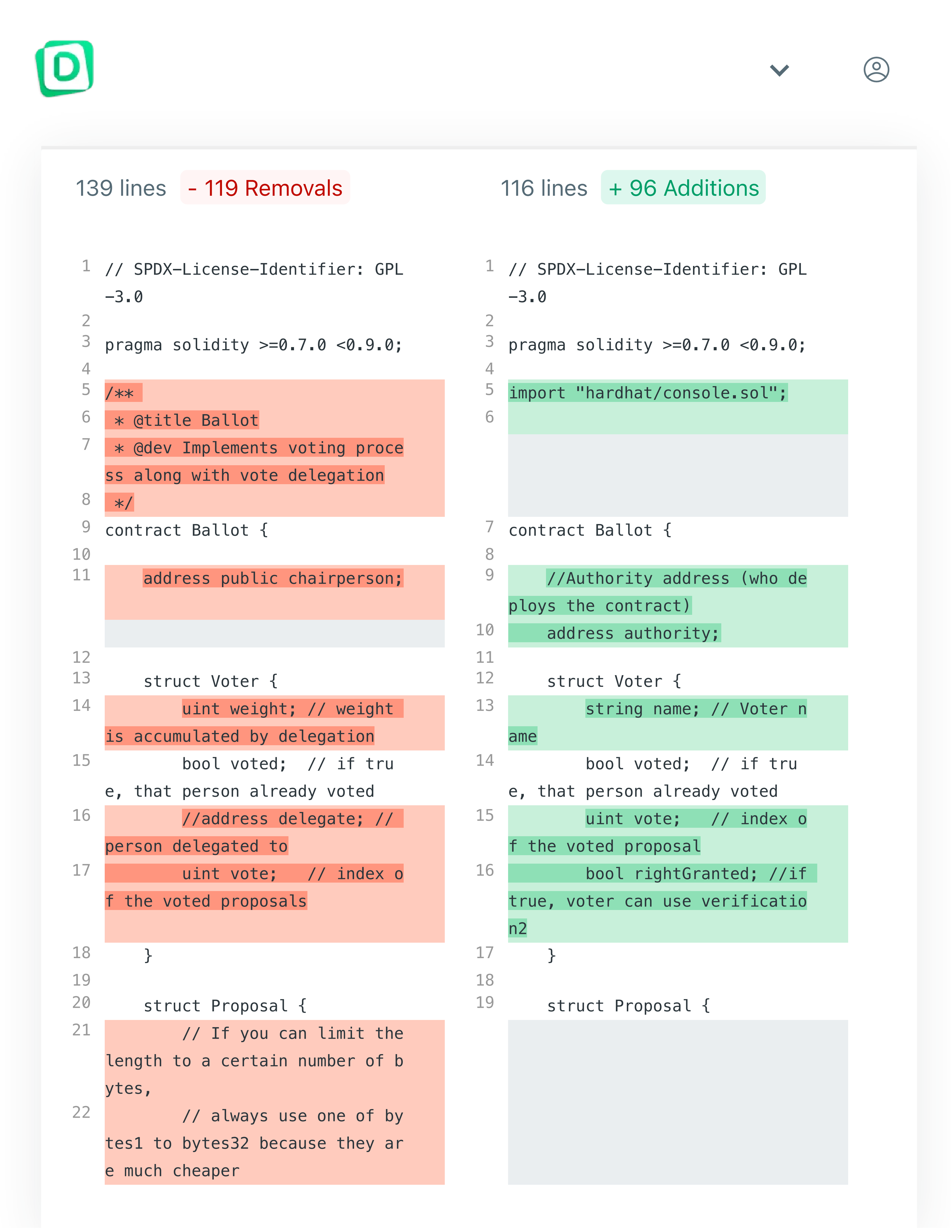

139 lines - 119 Removals

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Ballot
 * @dev Implements voting process along with vote delegation
 */
contract Ballot {

    address public chairperson;

    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        //address delegate; // person delegated to
        uint vote;   // index of the voted proposals

    }

    struct Proposal {
        // If you can limit the length to a certain number of bytes,
        // always use one of bytes1 to bytes32 because they are much cheaper
```

116 lines + 96 Additions

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

import "hardhat/console.sol";

contract Ballot {

    //Authority address (who deploys the contract)
    address authority;

    struct Voter {
        string name; // Voter name
        bool voted;  // if true, that person already voted
        uint vote;    // index of the voted proposal
        bool rightGranted; //if true, voter can use verification2
    }

    struct Proposal {
        string name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    mapping(address => Voter) public voters;

    Proposal[] public proposals;

    /**
     * @dev Create a new ballot to choose one of 'proposalNames'.
     * @param proposalNames names of proposals
     */
    constructor(string[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        for (uint i = 0; i < proposalNames.length; i++) {
            // 'Proposal({...})' creates a temporary
            // Proposal object and 'proposals.push(...)'
            // appends it to the end of 'proposals'.
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }

```

```
        string name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    mapping(address => Voter) voters;

    Proposal[] public proposals;

    constructor() {
        authority = msg.sender;

        //Appends two proposals to Proposal
        proposals.push(Proposal({ name: "Joe Biden", voteCount: 0 }));
        proposals.push(Proposal({ name: "Donald Trump", voteCount: 0 }));
    }

    /**
     * @dev Give 'voter' the right to vote on this ballot. May only be called by 'chairperson'.
     * @param voter address of voter
     */
    function giveRightToVote(address voter) public {
        require(
            msg.sender == chairperson,
            "Only chairperson can give right to vote."
        );
        require(
            !voters[voter].voted,
            "The voter already voted."
        );
        require(voters[voter].weight == 0);
        voters[voter].weight = 1;
    }

    /**
     * @dev Delegate your vote to the voter 'to'.
     * @param to address to which vote is delegated

```

```
    //Function to change name of voter
    function changeName(string memory _name, address _address) public{
            //Requirements
            require(msg.sender != authority, "Authority can not vote");
            require(msg.sender == _address, "Has no right to change name");

            //Change name
            voters[_address].name = _name;
    }

    //Function to give right to use function verification2
    function giveRightToVerification2(address _address) public {
            //Requirements
            require( msg.sender == authority, "Only authority can give right to watch" );

            //Grant permission
            voters[_address].rightGranted = true;
    }

    //Function to log out
    function logOut() public{
            voters[msg.sender].rightGranted = false;
    }
    function delegate(address to) public {
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You already voted.");
        require(to != msg.sender, "Self-delegation is disallowed.");

        while (voters[to].delegate != address(0)) {
            to = voters[to].delegate;

            // We found a loop in the delegation, not allowed.
            require(to != msg.sender, "Found loop in delegation.");
        }
        sender.voted = true;
        sender.delegate = to;
        Voter storage delegate_ = voters[to];
        if (delegate_.voted) {
            // If the delegate already voted,
```

```

    //Function to vote
    function vote(uint _proposal) public {
        Voter storage sender = voters[msg.sender];
        //Requirements
        require(msg.sender != authority, "Authority can not vote");
        require(bytes(sender.name).length != 0, "Please change your name first");
        require(!sender.voted, "Already voted");

        //Add vote
        sender.voted = true;
        sender.vote = _proposal;
        proposals[_proposal].voteCount += 1;
    }

    //Function to use Type 1 Verification
    function verification1(address _address, uint _token) public view returns(bool voted_){
        //Requirements
        require(msg.sender == _address, "Has no right to verify");
        require(_token == 123456 , "Invalid token");

        //Shows if user voted
        voted_ = voters[_address].voted;
```

```
            // directly add to the number of votes
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
            delegate_.weight += sender.weight;
        }
    }
    */

    /**
     * @dev Give your vote (including votes delegated to you) to proposal 'proposals[proposal].name'.
     * @param proposal index of proposal in the proposals array
     */
    function vote(uint proposal) public {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "Has no right to vote");
        require(!sender.voted, "Already voted.");
        sender.voted = true;
        sender.vote = proposal;

        // If 'proposal' is out of the range of the array,
        // this will throw automatically and revert all
```

```
    }

    //Function to use Type 2 Verification
    function verification2(address _address, uint _token) public view returns(string memory name_, bool voted_, string memory vote_){
        //Requirements
        require(msg.sender == _address, "Has no right to verify");
        require(_token == 123456 , "Invalid token");
        require(voters[_address].rightGranted == true, "Has no right granted");

        //Shows data voter
        name_ = voters[_address].name;
        // changes.
        proposals[proposal].voteCount += sender.weight;
    }

    /**
     * @dev Computes the winning proposal taking all previous votes into account.
     * @return winningProposal_ index of winning proposal in the proposals array
     */
    function winningProposal() public view
            returns (uint winningProposal_)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
```

```
        voted_ = voters[_address].voted;
        if (voted_ != false) {
            vote_ = proposals[voters[_address].vote].name;
        }
    }

    //Function to call winning proposal
    function winningProposal() public view returns (uint winningProposal_, string memory winnerName_) {
        //Winning proposal
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
        if (winningProposal_ == 0 ){
            winnerName_ = "NULL";
        } else {
            winnerName_ = proposals[winningProposal_].name;
        }
    }

    /**
     * @dev Calls winningProposal() function to get the index of the winner contained in the proposals array and then
     * @return winnerName_ the name of the winner
     */
    function winnerName() public view
            returns (string memory winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

```
    }

}
```