Assignment 1

```
Progma solidity ^0.8.0;
contract EscrowMarketplace {
     mapping(string => Item) public items;
   mapping(address => string[]) public buyerItems;
 mapping(address => string[]) public sellerItems;
     mapping(string => bool) public isItemSold;
    struct Item {
        string name;
        uint256 amount;
        address seller;
        address buyer;
        bool confirmed;
    }
event ItemListed(string indexed itemName, uint256 amount, address indexed seller);
    event ItemBought(string indexed itemName, uint256 amount, address indexed buyer);
    event ItemConfirmed(string indexed itemName, address indexed buyer);
    event FundsReleased(string indexed itemName, uint256 amount, address indexed seller);
    function listItem(string memory _itemName, uint256 _amount) public {
        require(bytes(_itemName).length > 0, "Item name cannot be empty");
        require(_amount > 0, "Item amount must be greater than 0");
        require(items[_itemName].amount == 0, "Item already listed");
        items[_itemName] = Item(_itemName, _amount, msg.sender, address(0), false);
```

```
        sellerItems[msg.sender].push(_itemName);
        emit ItemListed(_itemName, _amount, msg.sender);
    }
    function buy(string memory _itemName) public payable {
        require(bytes(_itemName).length > 0, "Item name cannot be empty");
        require(items[_itemName].amount > 0, "Item not listed");
        require(!isItemSold[_itemName], "Item already sold");
        require(msg.value >= items[_itemName].amount, "Insufficient funds");
        items[_itemName].buyer = msg.sender;
        isItemSold[_itemName] = true;
     buyerItems[msg.sender].push(_itemName);
        emit ItemBought(_itemName, items[_itemName].amount, msg.sender);
    }
    function confirmation(string memory _itemName) public {
        require(bytes(_itemName).length > 0, "Item name cannot be empty");
        require(items[_itemName].buyer == msg.sender, "Only buyer can confirm");
      require(!items[_itemName].confirmed, "Item already confirmed");
        items[_itemName].confirmed = true;
payable(items[_itemName].seller).transfer(items[_itemName].amount);
        emit ItemConfirmed(_itemName, msg.sender);
        emit FundsReleased(_itemName, items[_itemName].amount, items[_itemName].seller);
    }
    function disputeResolution(string memory _itemName) public {
```

```
    }
}
```

Assignment 2

```
pragma solidity ^0.8.0;
contract VotingSystem {
    mapping(address => bool) public registeredVoters;
    mapping(address => bool) public hasVoted;
    mapping(string => uint256) public voteCount;
    mapping(address => bool) public blacklistedVoters;
    mapping(address => string) public voterChoice;
    uint256 public registrationDeadline = 1626220800;
    event VoterRegistered(address indexed voter);
    event VoteCast(address indexed voter, string candidate);
    event VoterBlacklisted(address indexed voter);
    function register() public {
        require(block.timestamp < registrationDeadline, "Registration deadline has passed");
     require(!registeredVoters[msg.sender], "Voter already registered");
        registeredVoters[msg.sender] = true;
        emit VoterRegistered(msg.sender);
    }
```

```
    function castVote(string memory _candidate) public {
      require(registeredVoters[msg.sender], "Voter not registered");
       require(!blacklistedVoters[msg.sender], "Voter is blacklisted");
        if (hasVoted[msg.sender]) {
          voteCount[voterChoice[msg.sender]]--;
            blacklistedVoters[msg.sender] = true;
            emit VoterBlacklisted(msg.sender);
        }
        voteCount[_candidate]++;
        voterChoice[msg.sender] = _candidate;
        hasVoted[msg.sender] = true;
        emit VoteCast(msg.sender, _candidate);
    }
    function getVoteCount(string memory _candidate) public view returns (uint256) {
        return voteCount[_candidate];
    }
}
```