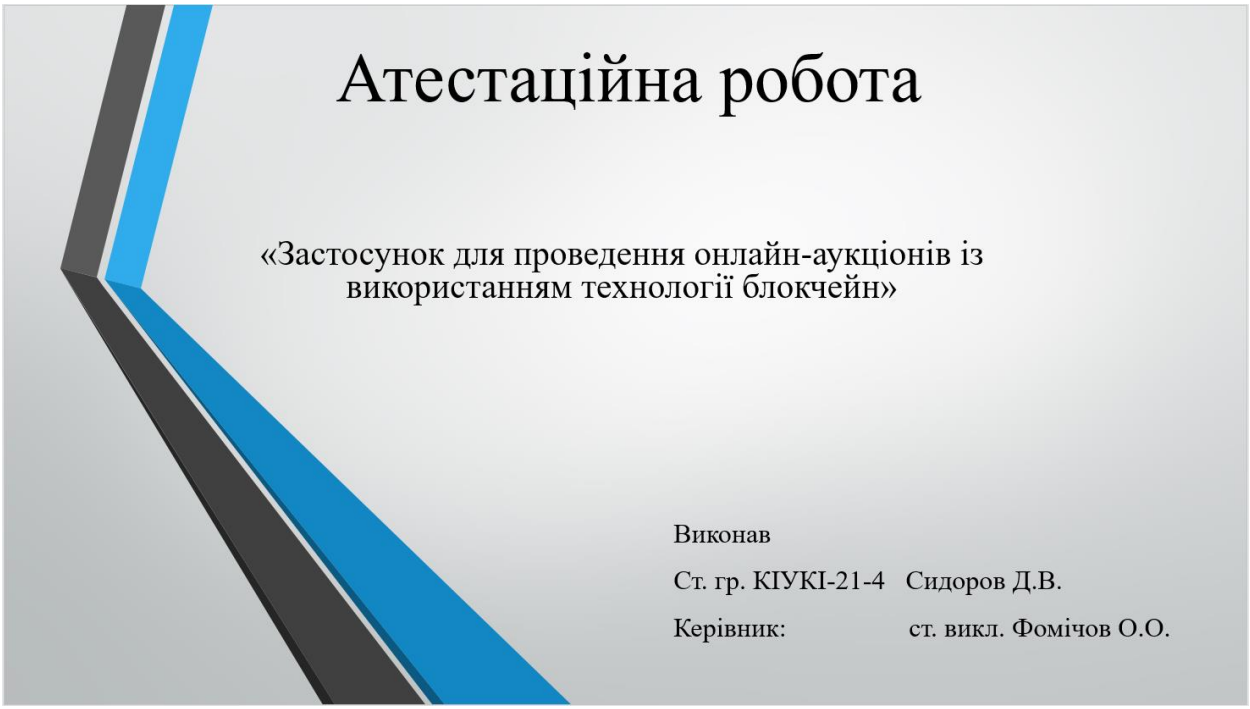
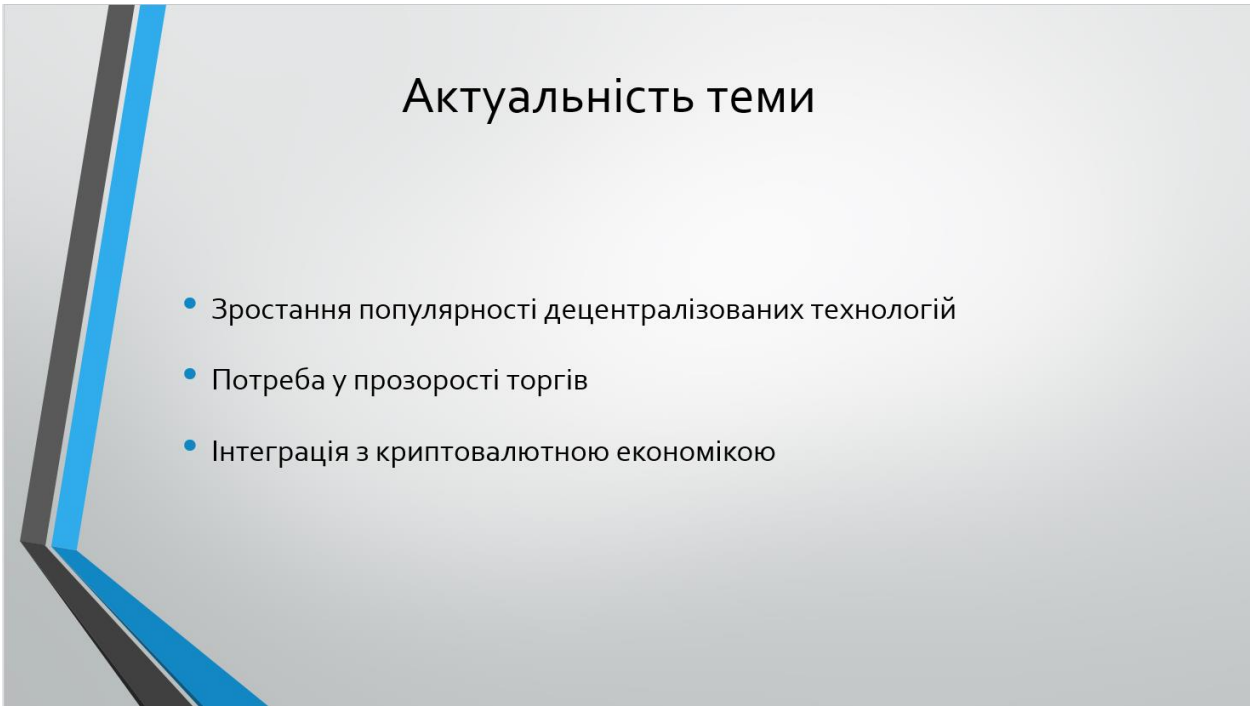
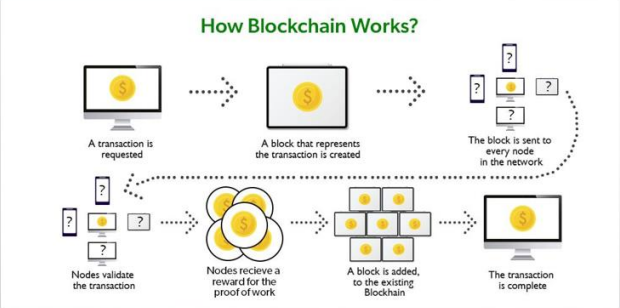
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи



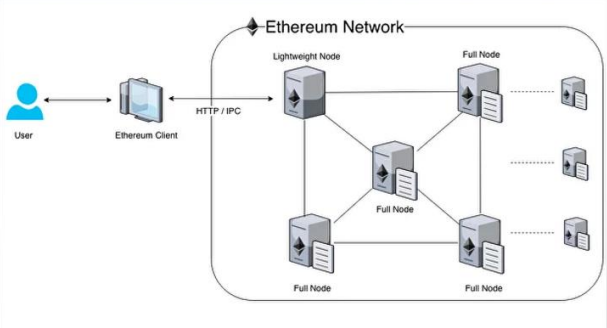


Відомості про блокчейн та як він працює



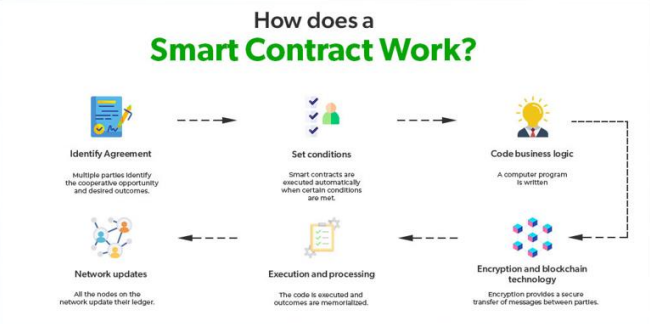
Блокчейн - це побудований за певними правилами безперервний ланцюжок блоків, зберігаючий деяку інформацію. Зв'язок між блоками забезпечується тим, що окрім нумірації кожен блок містить свою хеш-суму та хеш-суму попереднього блоку і тоді зміна інформації у блоці змінить його хеш-суму. Щоб виконалися правила побудови ланцюжка зміни хеш-суми потрібно записати до наступного блоку, що вже змінить його хеш-суму, при цьому попередні блоки будуть незмінні.

Блокчейн мережа Ethereum



Ethereum децентралізована блокчейн платформа, призначена для запуску смарт-контрактів та децентралізованих додатків. На відміну від біткойна, який орієнтований тільки на передачу цифрової валюти, Ethereum створенний як більш гнучка система, яка дозволяє програмувати умови передачі цієї валюти. Технічно платформа це глобальний децентралізований комп'ютер, який складається з тисяч вузлів кожен з яких зберігає та підтримує одну й туж саму копію блокчейна.

Смарт-контракти



Смарт-контракт – це код, записаний мовою програмування, який автоматично виконується в децентралізованому середовищі (наприклад, в мережі Ethereum). Після того, як смарт-контракт опубліковано в блокчейні, його не можна змінити — це забезпечує прозорість, довіру та стабільність умов угоди.
У буквальному сенсі, смарт-контракт — це цифровий аналог звичайного контракту, тільки замість паперової угоди й підписів сторін — машинний код і децентралізоване виконання.

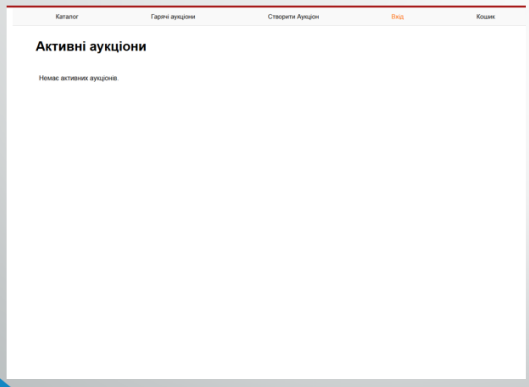
Застосування смарт-контракту у розробленому застосунку

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Auction {
    struct AuctionLot {
        address payable seller;
        string itemName;
        uint startPrice;
        uint highestBid;
        address payable highestBidder;
        uint endTime;
        bool ended;
        bool withdrawn;
    }

    uint public auctionCounter;
    mapping(uint => AuctionLot) public auctions;

    event AuctionCreated(uint auctionId, string itemName, uint endTime);
    event NewHighestBid(uint auctionId, address bidder, uint amount);
    event AuctionEnded(uint auctionId, address winner, uint amount);
    event Withdrawal(uint auctionId, address seller, uint amount);

    modifier onlyBeforeEnd(uint _auctionId) {
        require(block.timestamp < auctions[_auctionId].endTime, "Auction already ended");
        _;
    }

    modifier onlyAfterEnd(uint _auctionId) {
        require(block.timestamp >= auctions[_auctionId].endTime, "Auction not yet ended");
        _;
```

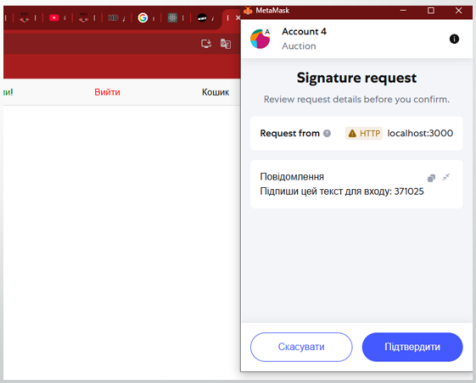
У розробленому застосунку смарт-контракт реалізує повний цикл проведення аукціону від створення до визначення переможця та переказ коштів. Уся логіка, що пов'язана зі ставками, перевіркою переможця та обробкою платежів, виконується напряму у блокчейні Ethereum. Такий підхід усуває ризики шахрайства, забезпечує довіру між сторонами і гарантує виконання усіх визначенних умов.

Демонстрація розробленого застосунку

Головне вікно сайту аукціону

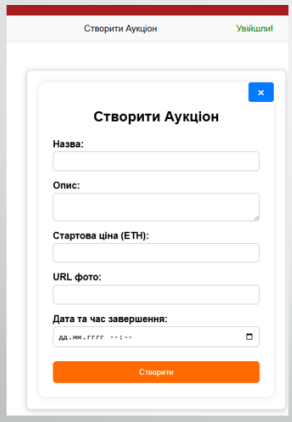


Підтвердження криптографічного підпису

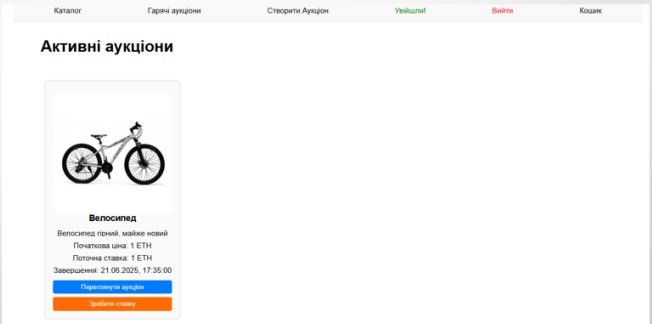


Демонстрація розробленого застосунку

Вікно створення нового лоту



Відображення створеного лоту на сторінці та можливі дії з ним



Можливі покращення та майбутнє проєкту

Що вже реалізовано	Майбутні покращення
Авторизація користувача за допомоги криптогаманця MetaMask	Покращення UI/UX дизайну сайту
Логіка проведення аукціонів	Додання нового функціоналу наприклад додання каталогу лотів та їх сортування
Можливість створювати нові лоти	Додання загальної бази даних
Взаємодія зі створеними лотами	Розгортання сайту на постійному хостингу

Висновки

- Проведено аналіз існуючих блокчейн мереж та мов програмування для смарт-контрактів
- Сформовано та реалізовано клієнт-серверну частину проєкту
- Реалізований повний процес проведення торгів на блокчейні

ДОДАТОК Б
ЛІСТИНГИ ПРОЄКТУ

Б.1 Лістинг файлу App.js

```
import React from "react";
import Header from "./Header";
import AuctionList from "./AuctionList";
import "./App.css";

function App() {
  return (
    <div>
      <Header />
      <main className="main-content">
        <h1 className="main-title">Активні аукціони</h1>
        <AuctionList />
      </main>
    </div>
  );
}

export default App;
```

Б.2 Лістинг файлу Header.js

```
import React, { useEffect, useState } from "react";
import { ethers } from "ethers";
import "./Header.css";
import CreateAuction from "./CreateAuction";

const Header = () => {
  const [userAddress, setUserAddress] = useState("");
  const [showCreateAuction, setShowCreateAuction] =
useState(false);
  const [status, setStatus] = useState("");

  // Перевіряємо токен при завантаженні
  useEffect(() => {
    const checkToken = async () => {
      const token = localStorage.getItem("token");
      const address = localStorage.getItem("userAddress");

      if (token && address) {
        try {
          const res = await
fetch("http://localhost:5000/api/check-token", {
            headers: { token },
          });
```

```
          if (!res.ok) throw new Error("Токен недійсний");
          const data = await res.json();

          if (data.success) {
            setUserAddress(address);
          } else {
            logout();
          }
        } catch (err) {
          console.warn("Помилка при перевірці токена:", err);
          logout();
        }
      }
    };

    checkToken();
  }, []);

  // Вхід через MetaMask
  const loginWithMetaMask = async () => {
    if (userAddress) return;

    if (!window.ethereum) {
      alert("Будь ласка, встановіть MetaMask!");
      return;
    }

    try {
      const provider = new
ethers.BrowserProvider(window.ethereum);
      await provider.send("eth_requestAccounts", []);
      const signer = await provider.getSigner();
      const address = await signer.getAddress();
      setUserAddress(address);
      localStorage.setItem("userAddress", address);

      const nonceRes = await
fetch(`http://localhost:5000/api/nonce?address=${address}`);
      if (!nonceRes.ok) throw new Error("Помилка отримання
nonce");
      const { nonce } = await nonceRes.json();

      const message = `Підпиши цей текст для входу: ${nonce}`;
      const signature = await signer.signMessage(message);

      const verifyRes = await
fetch("http://localhost:5000/api/verify-signature", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ address, signature, nonce }),
      });
```

```
      if (!verifyRes.ok) throw new Error("Помилка перевірки
підпису");

      const data = await verifyRes.json();

      if (data.success) {
        localStorage.setItem("token", data.token);
        alert("Успішно увійшли!");
      } else {
        alert("Помилка входу!");
        logout();
      }
    } catch (error) {
      console.error(error);
      alert("Помилка авторизації");
      logout();
    }
  };

  // Вихід
  const logout = () => {
    localStorage.removeItem("token");
    localStorage.removeItem("userAddress");
    setUserAddress("");
    alert("Ви вийшли з акаунту.");
  };

  return (
    <header className="header">
      <nav className="nav-menu">
        <span>Каталог</span>
        <span>Гарячі аукціони</span>
        <span
          style={{ cursor: "pointer" }}
          onClick={() => setShowCreateAuction(true)}
        >
          Створити Аукціон
        </span>
        {userAddress ? (
          <>
            <span style={{ color: "green" }}>Увійшли!</span>
            <span
              onClick={logout}
              style={{ cursor: "pointer", color: "red" }}
            >
              Вийти
            </span>
          </>
        ) : (
          <span
            onClick={loginWithMetaMask}
            style={{ cursor: "pointer", color: "#ff6b00" }}
          >
```

```
            Вхід
          </span>
        )}
        <span>Кошик</span>
      </nav>

      {status && (
        <p style={{ textAlign: "center", marginTop: "5px"
}}>{status}</p>
      )}

      {showCreateAuction && (
        <div className="auction-modal">
          <button
            onClick={() => setShowCreateAuction(false)}
            style={{ float: "right" }}
          >
            ✕
          </button>
          <CreateAuction />
        </div>
      )}
    </header>
  );
};

export default Header;
```

Б.3 Лістинг файлу CreateAuction.js

```
import React, { useState } from "react";
import { ethers } from "ethers";
import contractJSON from "./abi/auctionABI.json";
import "./CreateAuction.css";

const auctionABI = contractJSON.abi;
const CONTRACT_ADDRESS =
"0x5FbDB2315678afecb367f032d93F642f64180aa3";

const CreateAuction = () => {
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [startPrice, setStartPrice] = useState("");
  const [imageUrl, setImageUrl] = useState("");
  const [endTime, setEndTime] = useState("");
  const [status, setStatus] = useState("");

  const handleSubmit = async (e) => {
    e.preventDefault();

    const token = localStorage.getItem("token");
    if (!token) {
```

```
      setStatus("Будь ласка, увійдіть перед створенням
аукціону.");
      return;
    }

    try {
      if (!window.ethereum) {
        setStatus("Metamask не встановлено.");
        return;
      }

      setStatus("Очікуємо підтвердження транзакції...");

      const provider = new
ethers.BrowserProvider(window.ethereum);
      const accounts = await window.ethereum.request({ method:
"eth_requestAccounts" });
      const signer = await provider.getSigner(accounts[0]);

      const contract = new ethers.Contract(CONTRACT_ADDRESS,
auctionABI, signer);

      const currentTime = Math.floor(Date.now() / 1000);
      const endTimestamp = Math.floor(new
Date(endTime).getTime() / 1000);
      const biddingTime = endTimestamp - currentTime;

      if (biddingTime <= 0) {
        setStatus("Дата завершення має бути у майбутньому.");
        return;
      }

      const tx = await contract.createAuction(
        title,
        ethers.parseEther(startPrice),
        biddingTime
      );

      await tx.wait();

      const res = await fetch("http://localhost:5000/api/create-
auction", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          token,
        },
        body: JSON.stringify({
          title,
          description,
          startPrice,
          imageUrl,
          endTime,
```

```
        }),
      });

      const data = await res.json();
      if (data.success) {
        setStatus("Аукціон успішно створено!");
        setTitle("");
        setDescription("");
        setStartPrice("");
        setImageUrl("");
        setEndTime("");
      } else {
        setStatus("Помилка створення аукціону в базі даних.");
      }
    } catch (err) {
      console.error(err);
      setStatus("Сталася помилка під час створення аукціону.");
    }
  };

  return (
    <div className="create-auction-container">
      <h2>Створити Аукціон</h2>
      <form onSubmit={handleSubmit} className="create-auction-
form">
        <label>Назва:</label>
        <input
          type="text"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          required
        />

        <label>Опис:</label>
        <textarea
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          required
        />

        <label>Стартова ціна (ETH):</label>
        <input
          type="number"
          step="0.01"
          value={startPrice}
          onChange={(e) => setStartPrice(e.target.value)}
          required
        />

        <label>URL фото:</label>
        <input
          type="text"
          value={imageUrl}
```

```
          onChange={(e) => setImageUrl(e.target.value)}
        />

        <label>Дата та час завершення:</label>
        <input
          type="datetime-local"
          value={endTime}
          onChange={(e) => setEndTime(e.target.value)}
          required
        />

        <button type="submit">Створити</button>
      </form>
      {status && <p className="status-message">{status}</p>}
    </div>
  );
};

export default CreateAuction;
```

Б.4 Лістинг файлу AuctionList.js

```
import React, { useEffect, useState } from "react";
import { ethers } from "ethers";
import contractJSON from "./abi/auctionABI.json";
import "./AuctionList.css";

const auctionABI = contractJSON.abi;
const CONTRACT_ADDRESS =
"0x5FbDB2315678afecb367f032d93F642f64180aa3";

const AuctionList = () => {
  const [auctions, setAuctions] = useState([]);
  const [loading, setLoading] = useState(true);
  const [selectedAuction, setSelectedAuction] = useState(null);

  const fetchAuctions = async () => {
    try {
      const res = await
fetch("http://localhost:5000/api/auctions");
      const data = await res.json();
      setAuctions(data.auctions);
    } catch (error) {
      console.error("Помилка при завантаженні аукціонів:",
error);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchAuctions();
  }, []);
```

```
  const handleMakeBid = async (auctionId, currentBid) => {
    const token = localStorage.getItem("token");
    if (!token) {
      alert("Необхідно увійти для ставки.");
      return;
    }

    const amount = prompt("Введіть вашу ставку в ETH:");

    if (!amount || isNaN(amount)) {
      alert("Некоректна ставка");
      return;
    }

    try {
      if (!window.ethereum) {
        alert("Потрібен Metamask");
        return;
      }

      const provider = new
ethers.BrowserProvider(window.ethereum);
      const signer = await provider.getSigner(); // 🛠️ НЕ
передаємо адресу
      const contract = new ethers.Contract(CONTRACT_ADDRESS,
auctionABI, signer);

      const parsedAmount = ethers.parseEther(amount);

      const tx = await contract.bid(auctionId, { value:
parsedAmount });
      await tx.wait();

      // Оновлення бекенду
      await fetch(`http://localhost:5000/api/place-
bid/${auctionId}`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          token,
        },
        body: JSON.stringify({ bidAmount: parseFloat(amount) }),
      });

      alert("Ставка зроблена успішно!");
      fetchAuctions(); // оновлення списку
    } catch (err) {
      console.error(err);
      alert("Помилка при ставці: " + (err?.message ||
"невідома"));
    }
  };
```

```
  return (
    <div className="auction-list-container">
      {loading ? (
        <p>Завантаження...</p>
      ) : auctions.length === 0 ? (
        <p>Немає активних аукціонів.</p>
      ) : (
        <div className="auction-list">
          {auctions.map((auction) => (

            <div
              key={auction.id}

              className={`auction-card ${auction.ended ? "ended"
: ""}`}
            >
              {auction.imageUrl && (
                <img
                  src={auction.imageUrl}
                  alt={auction.title}
                  style={{
                    width: "100%",
                    borderRadius: "8px",
                    marginTop: "10px",
                  }}
                />
              )}
              <h3>{auction.title}</h3>
              <p
className="description">{auction.description}</p>
              <p>Початкова ціна: {auction.startPrice} ETH</p>
              <p>Поточна ставка: {auction.currentBid ||
auction.startPrice} ETH</p>
              <p>Завершення: {new
Date(auction.endTime).toLocaleString()}</p>

              <div className="button-group">
                <button
                  className="view-button"
                  onClick={() => setSelectedAuction(auction)}
                >
                  Переглянути аукціон
                </button>
                <button
                  disabled={auction.ended}
                  onClick={() => handleMakeBid(auction.id,
auction.currentBid)}
                >
                  Зробити ставку
                </button>
              </div>
            </div>
```

```
          ))}
        </div>
      )}

      {selectedAuction && (
        <div className="modal">
          <div className="modal-content">
            <button
              className="close-button"
              onClick={() => setSelectedAuction(null)}
            >
              ×
            </button>
            <h2>{selectedAuction.title}</h2>
            <p><strong>Опис:</strong>
{selectedAuction.description}</p>
            <p><strong>Початкова ціна:</strong>
{selectedAuction.startPrice} ETH</p>
            <p><strong>Поточна ставка:</strong>
{selectedAuction.currentBid || selectedAuction.startPrice}
ETH</p>
            <p><strong>Завершення:</strong> {new
Date(selectedAuction.endTime).toLocaleString()}</p>
            <p><strong>Власник:</strong>
{selectedAuction.owner}</p>
            {selectedAuction.imageUrl && (
              <img
                src={selectedAuction.imageUrl}
                alt={selectedAuction.title}
                style={{ width: "100%", borderRadius: "8px",
marginTop: "10px" }}
              />
            )}
            {selectedAuction.ended && (
              <p style={{ color: "red", fontWeight: "bold"
}}>Аукціон завершено</p>
            )}
          </div>
        </div>
      )}
    </div>
  );
};

export default AuctionList;
```

Б.5 Лістинг файлу index.js

```
const path = require("path");
require("dotenv").config({ path:
path.resolve("E:/VSCodeProjects/Auction/backend", "ini.env") });
const express = require("express");
```

```
const cors = require("cors");
const bodyParser = require("body-parser");
const jwt = require("jsonwebtoken");
const { verifyMessage } = require("ethers");
const cron = require("node-cron");
const auctionContract = require("./contract");
const { getNonce, setNonce, generateNonce } =
require("./noncestore");

const app = express();
const PORT = 5000;

app.use(cors());
app.use(bodyParser.json());

const auctions = []; // тимчасове сховище

// ------------------- NONCE -------------------

app.get("/api/nonce", (req, res) => {
  const { address } = req.query;
  if (!address) return res.status(400).json({ error: "Address is
required" });
  const nonce = getNonce(address);
  res.json({ nonce });
});

// ------------------- SIGNATURE -------------------

app.post("/api/verify-signature", async (req, res) => {
  const { address, signature, nonce } = req.body;
  if (!address || !signature || !nonce)
    return res.status(400).json({ error: "Missing data" });

  try {
    const message = `Підпиши цей текст для входу: ${nonce}`;
    const signerAddress = await verifyMessage(message,
signature);

    if (signerAddress.toLowerCase() === address.toLowerCase()) {
      setNonce(address, generateNonce());
      const token = jwt.sign({ address },
process.env.JWT_SECRET, { expiresIn: "1h" });
      res.json({ success: true, token });
    } else {
      res.status(401).json({ error: "Invalid signature" });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Server error" });
  }
});
```

```
// ------------------- TOKEN CHECK -------------------

app.get("/api/check-token", (req, res) => {
  const { token } = req.headers;
  if (!token) return res.status(401).json({ success: false,
error: "No token provided" });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    res.json({ success: true, address: decoded.address });
  } catch (err) {
    console.error(err);
    res.status(401).json({ success: false, error: "Invalid
token" });
  }
});

// ------------------- AUCTIONS -------------------

app.post("/api/create-auction", (req, res) => {
  const { token } = req.headers;
  const { title, description, startPrice, imageUrl, endTime } =
req.body;

  if (!token) return res.status(401).json({ error: "No token
provided" });
  if (!endTime) return res.status(400).json({ error: "End time
is required" });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const newAuction = {
      id: auctions.length + 1,
      title,
      description,
      startPrice: parseFloat(startPrice),
      imageUrl,
      endTime: new Date(endTime),
      owner: decoded.address,
      createdAt: new Date(),
      ended: false,
    };
    auctions.push(newAuction);
    res.json({ success: true, auction: newAuction });
  } catch (err) {
    console.error(err);
    res.status(401).json({ error: "Invalid token" });
  }
});

app.get("/api/auctions", (req, res) => {
  res.json({ auctions });
});
```

```
app.post("/api/place-bid/:id", (req, res) => {
  const { id } = req.params;
  const { token } = req.headers;
  const { bidAmount } = req.body;

  if (!token) return res.status(401).json({ success: false,
error: "No token provided" });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const auction = auctions.find((a) => a.id === parseInt(id));

    if (!auction) return res.status(404).json({ success: false,
error: "Аукціон не знайдено" });
    if (auction.ended) return res.status(400).json({ success:
false, error: "Аукціон вже завершено" });

    auction.currentBid = parseFloat(bidAmount);
    auction.highestBidder = decoded.address;
    auction.bids = auction.bids || [];
    auction.bids.push({
      amount: auction.currentBid,
      bidder: auction.highestBidder,
      time: new Date(),
    });

    res.json({
      success: true,
      message: "Ставка збережена в локальному сховищі",
      auction,
    });
  } catch (err) {
    console.error(err);
    res.status(401).json({ success: false, error: "Invalid
token" });
  }
});

// ------------------- CRON: Автоматичне завершення ------------
-------

cron.schedule("*/30 * * * * *", async () => {
  const now = new Date();

  for (let auction of auctions) {
    if (!auction.ended && new Date(auction.endTime) <= now) {
      try {
        const tx = await auctionContract.endAuction(auction.id);
        await tx.wait();
        auction.ended = true;
        console.log(`✅ Аукціон ${auction.id} завершено на
блокчейні`);
```

```
      } catch (err) {
        console.error(`✖ Помилка завершення аукціону
${auction.id}:`, err.message);
      }
    }
  }
});

// ------------------- START -------------------

app.listen(PORT, () => console.log(`Server running on port
${PORT}`));
```

Б.6 Лістинг файлу Auction.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Auction {
    struct AuctionLot {
        address payable seller;
        string itemName;
        uint startPrice;
        uint highestBid;
        address payable highestBidder;
        uint endTime;
        bool ended;
        bool withdrawn;              // Чи вже виведені кошти
    }

    uint public auctionCounter;
    mapping(uint => AuctionLot) public auctions;

    event AuctionCreated(uint auctionId, string itemName, uint
endTime);
    event NewHighestBid(uint auctionId, address bidder, uint
amount);
    event AuctionEnded(uint auctionId, address winner, uint
amount);
    event Withdrawal(uint auctionId, address seller, uint
amount);

    modifier onlyBeforeEnd(uint _auctionId) {
        require(block.timestamp < auctions[_auctionId].endTime,
"Auction already ended");
        _;
    }

    modifier onlyAfterEnd(uint _auctionId) {
        require(block.timestamp >= auctions[_auctionId].endTime,
"Auction not yet ended");
        _;
```

```
    }

    function createAuction(
        string memory _itemName,
        uint _startingPrice,
        uint _biddingTimeInSeconds
    ) external {
        require(_startingPrice > 0, "Starting price must be
positive");
        require(_biddingTimeInSeconds > 0, "Bidding time must be
positive");

        auctionCounter++;
        auctions[auctionCounter] = AuctionLot({
            seller: payable(msg.sender),
            itemName: _itemName,
            startPrice: _startingPrice,
            highestBid: _startingPrice,
            highestBidder: payable(address(0)),
            endTime: block.timestamp + _biddingTimeInSeconds,
            ended: false,
            withdrawn: false
        });

        emit AuctionCreated(auctionCounter, _itemName,
block.timestamp + _biddingTimeInSeconds);
    }

    function bid(uint _auctionId) external payable
onlyBeforeEnd(_auctionId) {
        AuctionLot storage auction = auctions[_auctionId];
        require(msg.sender != auction.seller, "Seller cannot
bid");
        require(msg.value > auction.highestBid, "Bid must be
higher than current highest bid");

        // Повернення попередньої ставки, якщо є попередній
учасник
        if (auction.highestBidder != address(0)) {
            (bool refundSuccess, ) =
auction.highestBidder.call{value: auction.highestBid}("");
            require(refundSuccess, "Failed to refund previous
highest bidder");
        }

        auction.highestBidder = payable(msg.sender);
        auction.highestBid = msg.value;

        emit NewHighestBid(_auctionId, msg.sender, msg.value);
    }

    function endAuction(uint _auctionId) external
onlyAfterEnd(_auctionId) {
```

```
        AuctionLot storage auction = auctions[_auctionId];
        require(!auction.ended, "Auction already ended");

        auction.ended = true;

        if (auction.highestBidder != address(0)) {
            (bool success, ) = auction.seller.call{value:
auction.highestBid}("");
            require(success, "Failed to send funds to seller");

            auction.withdrawn = true;
            emit Withdrawal(_auctionId, auction.seller,
auction.highestBid);
        }

        emit AuctionEnded(_auctionId, auction.highestBidder,
auction.highestBid);
    }

    function getAuction(uint _auctionId) external view returns (
        address seller,
        string memory itemName,
        uint startPrice,
        uint highestBid,
        address highestBidder,
        uint endTime,
        bool ended,
        bool withdrawn
    ) {
        AuctionLot memory auction = auctions[_auctionId];
        return (
            auction.seller,
            auction.itemName,
            auction.startPrice,
            auction.highestBid,
            auction.highestBidder,
            auction.endTime,
            auction.ended,
            auction.withdrawn
        );
    }
}
```