BlockChain

"Aplikasi BlockChain Chat "

Dosen pengampu

Ade Kurniawan, S.Pd., M.Pd.T

Di susun oleh:

TAUFIQUL ISRAT 22346026

FAKULTAS TEKNIK DEPARTEMEN ELEKTRONIKA

PRODI INFORMATIKA UNIVERSITAS NEGERI

PADANG

1. Deskripsi Umum

Aplikasi ini merupakan prototipe sistem chatting dua arah berbasis blockchain yang memanfaatkan Ethereum smart contract, MetaMask untuk otentikasi akun pengguna, dan React.js untuk antarmuka pengguna. Setiap pesan yang dikirim akan dicatat ke dalam blockchain lokal (via Hardhat) dan bisa dibaca oleh kedua pihak pengirim dan penerima.

2. Teknologi yang Digunakan

• Blockchain: Ethereum (via Hardhat local node)
• Bahasa Smart Contract: Solidity
• Frontend: React.js
• Wallet & Akses: MetaMask (melalui ethers.js)
• Library Web3: ethers.js
• CSS Styling: Custom CSS (responsive & modern UI)
• Local Deployment: Hardhat (network: localhost:8545)

3. Fitur Utama

• Koneksi MetaMask dan pengenalan akun
• Kirim pesan ke alamat Ethereum lain
• Ambil seluruh pesan berdasarkan dua alamat wallet
• UI yang membedakan antara pesan yang dikirim dan diterima
• Pembaruan pesan otomatis tiap detik (real-time polling)

4. Struktur Smart Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract ChatApp {
    struct Message {
        address from;
        address to;
        string content;
        uint timestamp;
    }
    Message[] public messages;
    event MessageSent(address indexed from, address indexed to, string content, uint
timestamp);
    function sendMessage(address _to, string memory _content) public {
        messages.push(Message(msg.sender, _to, _content, block.timestamp));
        emit MessageSent(msg.sender, _to, _content, block.timestamp);
    }
    function getMessages(address user1, address user2) public view returns (Message[]
memory) {
        uint count;
        for (uint i = 0; i < messages.length; i++) {
            if (
                (messages[i].from == user1 && messages[i].to == user2) ||
                (messages[i].from == user2 && messages[i].to == user1)
            ) {
                count++;
            }
        }
        Message[] memory result = new Message[](count);
        uint index;
        for (uint i = 0; i < messages.length; i++) {
            if (
                (messages[i].from == user1 && messages[i].to == user2) ||
                (messages[i].from == user2 && messages[i].to == user1)
            ) {
                result[index] = messages[i];
                index++;
            }
        }
```

```
        return result;
    }
}
```

5. Script Deploy Smart Contract

```
const hre = require("hardhat");

async function main() {
  const ChatApp = await hre.ethers.getContractFactory("ChatApp");
  const chatApp = await ChatApp.deploy();

  await chatApp.waitForDeployment(); // gunakan ini di Hardhat terbaru
  console.log("ChatApp deployed to:", await chatApp.getAddress());
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

6. Frontend: React (App.js)

• Koneksi ke MetaMask
• Pemilihan penerima berdasarkan alamat Ethereum
• Input dan kirim pesan ke kontrak
• Ambil pesan dari blockchain dengan getMessages(user1, user2)
• Tampilkan pesan di UI sesuai pengirim

```
import React, { useState, useEffect, useRef } from "react";
import { ethers } from "ethers";
import ChatAppABI from "./ChatAppABI.json";
import "./App.css";

const contractAddress = "0x5FbDB2315678afecb367f032d93F642f64180aa3";

function App() {
  const [provider, setProvider] = useState(null);
  const [signer, setSigner] = useState(null);
  const [contract, setContract] = useState(null);
  const [account, setAccount] = useState("");
  const [recipient, setRecipient] = useState("");
  const [message, setMessage] = useState("");
```

```
const [messages, setMessages] = useState([]);
const [connecting, setConnecting] = useState(false);
// const messagesEndRef = useRef(null);  // tidak perlu kalau gak scroll otomatis

useEffect(() => {
  if (window.ethereum) {
    const prov = new ethers.BrowserProvider(window.ethereum);
    setProvider(prov);
  } else {
    alert("Please install MetaMask!");
  }
}, []);

async function connectWallet() {
  if (!provider || connecting) return;
  setConnecting(true);
  try {
    const accounts = await window.ethereum.request({ method: "eth_accounts" });
    const activeAccount =
      accounts.length > 0
        ? accounts[0]
        : (await window.ethereum.request({ method: "eth_requestAccounts" }))[0];
    const signer = await provider.getSigner();
    const chatContract = new ethers.Contract(contractAddress, ChatAppABI, signer);
    setSigner(signer);
    setAccount(activeAccount);
    setContract(chatContract);
  } catch (err) {
    alert("Failed to connect: " + err.message);
  } finally {
    setConnecting(false);
  }
}

async function sendMessage() {
  if (!contract) return;
  if (!ethers.isAddress(recipient)) return alert("Invalid recipient address");
  if (!message.trim()) return alert("Message cannot be empty");
  try {
    const tx = await contract.sendMessage(recipient, message);
    await tx.wait();
    setMessage("");
    fetchIncomingMessages();
  } catch (err) {
    alert("Error sending message: " + err.message);
```

```
  }
}

async function fetchIncomingMessages() {
  if (!contract || !account || !recipient) return;
  try {
    const incomingMsgs = await contract.getMessages(recipient, account);
    setMessages(incomingMsgs);
  } catch (err) {
    console.error("Fetch incoming messages error:", err);
  }
}

useEffect(() => {
  if (!account || !recipient) return;

  fetchIncomingMessages();
  const interval = setInterval(fetchIncomingMessages, 1000);
  return () => clearInterval(interval);
}, [account, recipient]);

// Hapus useEffect scroll otomatis, jadi scroll posisi chat tetap manual user

return (
  <div className="app-container">
    <h1>Blockchain Chat</h1>
    {!account ? (
      <button onClick={connectWallet} disabled={connecting}>
        {connecting ? "Connecting..." : "Connect MetaMask"}
      </button>
    ) : (
      <div>
        <p className="account-info">
          <b>Connected:</b> {account}
        </p>
        <input
          type="text"
          placeholder="Recipient public address"
          value={recipient}
          onChange={(e) => setRecipient(e.target.value)}
          style={{ width: "100%", padding: "8px" }}
        />
        <textarea
          placeholder="Type your message here"
          value={message}
```

```
          onChange={(e) => setMessage(e.target.value)}
          rows={4}
          cols={50}
          style={{ marginTop: 10, width: "100%", padding: "8px" }}
        />
        <button onClick={sendMessage} style={{ marginTop: 10 }}>
          Send Message
        </button>
        <hr style={{ margin: "20px 0", borderColor: "#764ba2" }} />
        <h3>Incoming Messages from Recipient</h3>
        <div
          className="messages-container"
          style={{ maxHeight: "300px", overflowY: "auto" }}
        >
          {messages.length === 0 && <p>No incoming messages yet</p>}
          {messages.map((msg, idx) => {
            const isOwn = msg.from.toLowerCase() === account.toLowerCase();
            return (
              <div key={idx} className={`message ${isOwn ? "sent" : "received"}`}>
                <p>
                  <b>{isOwn ? "You" : "Them"}:</b> {msg.content}
                </p>
                <small>
                  <i>{new Date(Number(msg.timestamp) * 1000).toLocaleString()}</i>
                </small>
              </div>
            );
          })}
        </div>
      </div>
    )}
  </div>
 );
}

export default App;
```

7. Styling (App.css)

• Desain UI modern dan gelap
• Bubble chat untuk pengiriman dan penerimaan pesan
• Respon antarmuka untuk perangkat mobile
• Scroll bar custom pada tampilan pesan

```
/* Reset dasar */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

/* Body background */
body {
  background: linear-gradient(135deg, #667eea, #4b95a2);
  color: #fff;
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: flex-start;
  padding: 40px 20px;
}

/* Container utama */
.app-container {
  background: #1a1a2e;
  border-radius: 16px;
  box-shadow: 0 8px 24px rgba(102, 126, 234, 0.3);
  max-width: 600px;
  width: 100%;
  padding: 30px 40px;
}

/* Judul */
h1 {
  font-weight: 700;
  font-size: 2.4rem;
  margin-bottom: 24px;
  text-align: center;
  letter-spacing: 1.2px;
  color: #e0e0ff;
}

/* Tombol */
button {
  background: #4b9ca2;
  color: #fff;
  border: none;
  padding: 12px 26px;
```

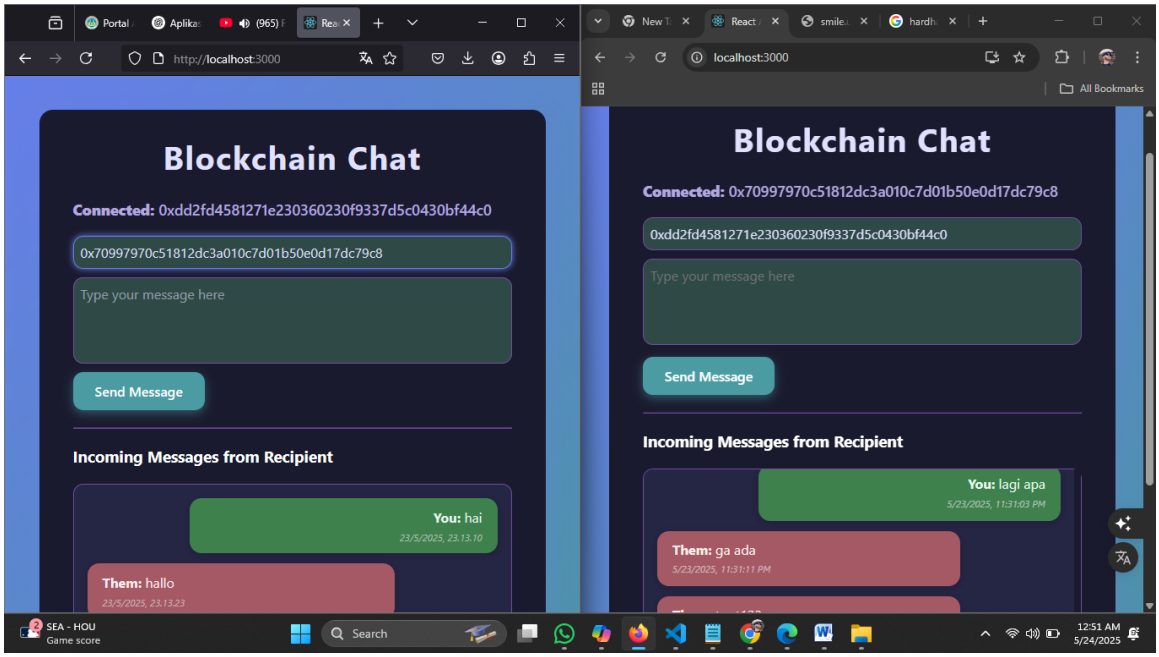
```
  font-size: 1rem;
  border-radius: 12px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
  box-shadow: 0 4px 12px rgba(75, 156, 162, 0.5);
}

button:hover {
  background: #667eea;
  box-shadow: 0 6px 20px rgba(102, 126, 234, 0.7);
}

/* Input dan textarea */
input[type="text"], textarea {
  background: #2f4a46;
  border: 1.5px solid #764ba2;
  border-radius: 12px;
  padding: 12px 16px;
  font-size: 1rem;
  color: #e0e0ff;
  width: 100%;
  outline: none;
  transition: border-color 0.3s ease;
  resize: none;
  font-family: inherit;
}

input[type="text"]:focus, textarea:focus {
  border-color: #667eea;
  box-shadow: 0 0 8px #667eea;
}

/* Connected account info */
.account-info {
  font-weight: 600;
  margin-bottom: 20px;
  font-size: 1.1rem;
  word-break: break-all;
  color: #b3a7e1;
}

/* Messages container */
.messages-container {
  max-height: 320px;
```

```
  overflow-y: auto;
  border: 1.5px solid #764ba2;
  border-radius: 12px;
  padding: 16px;
  background: #252544;
  margin-top: 20px;
}

/* Message bubble */
.message {
  padding: 12px 18px;
  border-radius: 14px;
  margin-bottom: 12px;
  max-width: 75%;
  position: relative;
  word-wrap: break-word;
  font-size: 1rem;
  line-height: 1.3;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.15);
}

.message.sent {
  background: #4caf50aa;
  color: #e9f7ef;
  margin-left: auto;
  text-align: right;
}

.message.received {
  background: #e57373aa;
  color: #fff0f0;
  margin-right: auto;
  text-align: left;
}

/* Message metadata */
.message small {
  display: block;
  margin-top: 6px;
  font-size: 0.7rem;
  opacity: 0.7;
  color: #ddd;
}

/* Scrollbar styling */
```

```
.messages-container::-webkit-scrollbar {
  width: 8px;
}

.messages-container::-webkit-scrollbar-thumb {
  background: #764ba2;
  border-radius: 8px;
}

.messages-container::-webkit-scrollbar-track {
  background: #1a1a2e;
}

/* Responsive */
@media (max-width: 650px) {
  .app-container {
    padding: 20px;
  }
  input[type="text"], textarea {
    font-size: 0.9rem;
  }
  button {
    width: 100%;
  }
}
```

8. Testing dan Pengujian

• Dua browser digunakan dengan dua akun MetaMask
• Hardhat dijalankan: npx hardhat node
• Deploy kontrak: npx hardhat run scripts/deploy.js --network localhost
• Jalankan React app: npm start

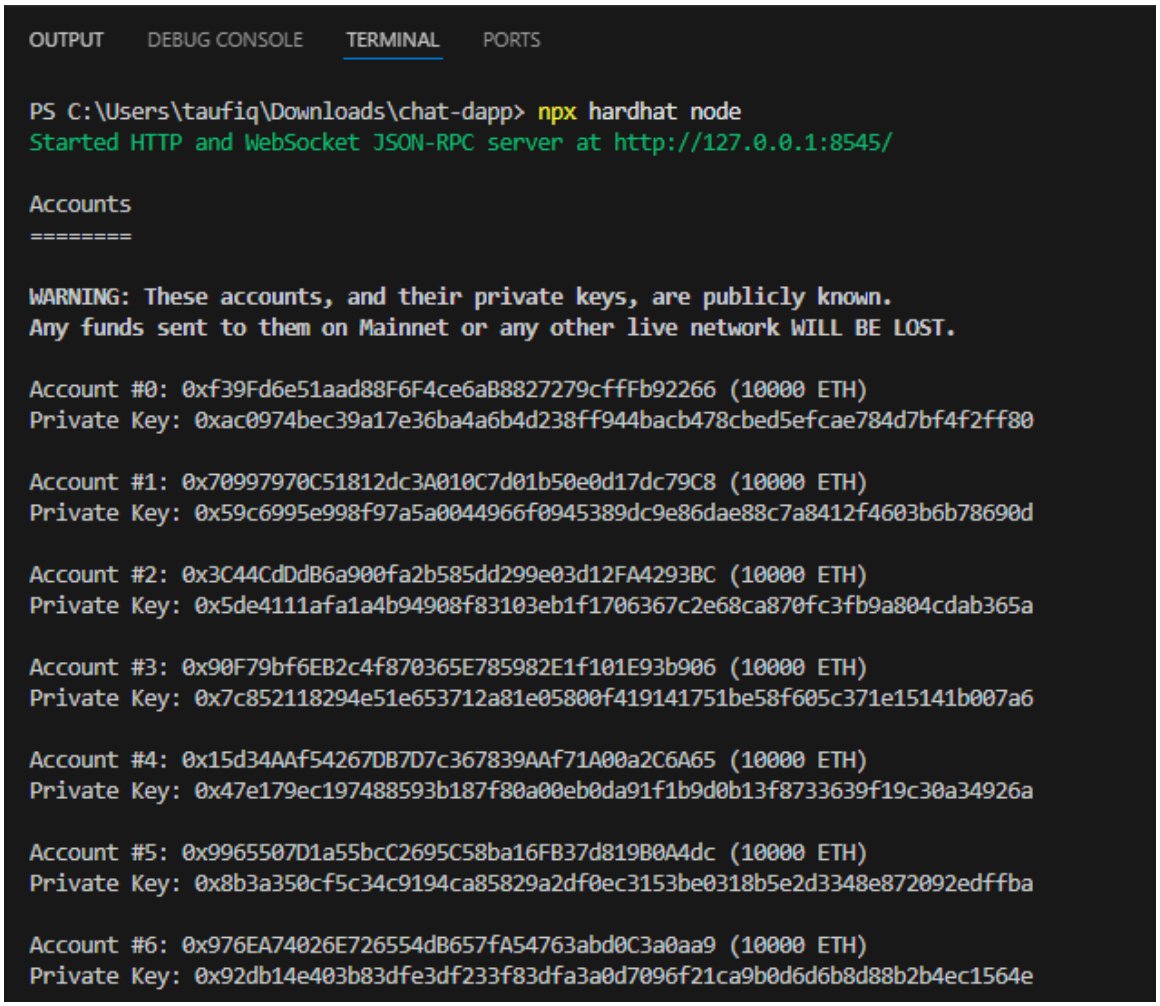


9. Kelebihan Sistem

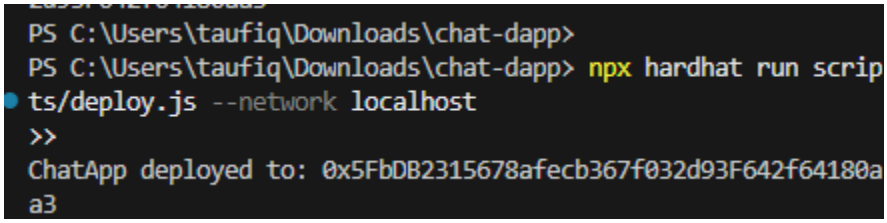
- Pesan disimpan di blockchain dan tidak bisa diubah/hapus
- Antarmuka sederhana dan mudah digunakan
- Tidak membutuhkan backend server
- Real-time polling untuk pembaruan pesan

Cara menggunakanya

1. Jalankan npx hardhat node untuk mendapatkan akun masing” akun ada 10000 ETH



2. Jalankannpx hardhat run scripts/deploy.js --network localhost untuk mendapatkan contract addres di terminal baru



3. Masuk ke PS C:\Users\taufiq\Downloads\chat-dapp\frontend> lalu ketik npm start
   Maka anda akan langsung di alihkan ke browser ,buka di 2 brwser lalu hubungkan keduanya ke metamask masukan alamat penerimanya