BLOCKCHAIN-BASED SOCIAL MEDIA PLATFORM FOR IMMUTABLE IMAGE SHARING

Oleh:
Aditias Zulmatoriq (F1D02310031)[1],
Dodi Wijaya (F1D02310047)[2],
Muhammad Rendi Maulana (F1D02310081)[3]

Program Studi Teknik Informatika, Fakultas Teknik,
Universitas Mataram
2025

Abstrak

Perkembangan teknologi digital memicu semakin tingginya kebutuhan akan platform berbagi konten visual yang aman, transparan, dan terlindungi dari manipulasi. Platform media sosial konvensional masih bergantung pada arsitektur terpusat yang rentan terhadap sensor, penghapusan sepihak oleh penyedia layanan, serta risiko kebocoran data. Penelitian ini mengusulkan pengembangan platform media sosial berbasis blockchain yang memungkinkan pengguna mengunggah dan membagikan gambar secara immutable melalui integrasi IPFS sebagai penyimpanan terdesentralisasi dan Ethereum Smart Contract sebagai pencatat metadata permanen. Sistem dikembangkan menggunakan React untuk antarmuka pengguna, MetaMask untuk otentikasi akun blockchain, dan Ether.js sebagai library penghubung frontend–blockchain. Hasil implementasi menunjukkan bahwa platform berhasil menyimpan gambar di IPFS secara permanen dan mencatat metadata postingan—termasuk CID, pembuat, timestamp, likes, serta komentar—langsung pada smart contract. Pendekatan ini memperkuat aspek keamanan, transparansi, serta melindungi hak pengguna atas data mereka.
Kata kunci: Blockchain, IPFS, Smart Contract, React, Media Sosial Terdesentralisasi.

1. PENDAHULUAN

Platform media sosial berperan penting dalam penyebaran informasi dan konten visual, namun sebagian besar masih menggunakan arsitektur client–server sehingga seluruh data berada di bawah kendali penyedia platform. Model ini memiliki kekurangan utama, seperti:

1. Single Point of Failure, serangan terhadap server pusat dapat menyebabkan hilangnya seluruh layanan.
2. Manipulasi dan Penghapusan Data, penyedia platform dapat menghapus konten tanpa persetujuan pengguna.
3. Risiko Privasi, data pengguna dapat disalahgunakan karena tersimpan secara terpusat.

Zyskind & Nathan [1] menegaskan bahwa arsitektur terpusat secara inheren rentan terhadap penyalahgunaan data dan pelanggaran privasi. Di sisi lain, blockchain menyediakan mekanisme pencatatan data yang immutable, terverifikasi, dan transparan, sehingga dapat menjadi fondasi alternatif untuk platform media sosial. Teknologi InterPlanetary File System (IPFS) menambahkan lapisan penyimpanan off-chain yang terdesentralisasi dan efisien. Penelitian oleh Li et al. [2]

menunjukkan bahwa penggabungan blockchain dan IPFS meningkatkan integritas data serta memberikan mekanisme verifikasi content-based melalui CID.

Berdasarkan kondisi tersebut, penelitian ini mengembangkan decentralized social media platform yang memungkinkan pengguna mengunggah gambar, memberikan like, dan menambah komentar melalui smart contract Ethereum dan IPFS/Pinata.

1.1 Rumusan Masalah

Penelitian ini dirumuskan ke dalam dua permasalahan utama:

1. Bagaimana mengatasi risiko penyalahgunaan data pada platform media sosial terpusat dengan menghadirkan mekanisme berbagi gambar yang tidak dapat dimanipulasi?
2. Bagaimana merancang dan membangun sistem berbagi gambar berbasis blockchain dengan integrasi IPFS, React, MetaMask, dan Ether.js?

1.2 Tujuan Penelitian

Tujuan penelitian ini adalah untuk:

1. Membangun platform berbagi gambar berbasis blockchain yang aman dan terdesentralisasi.
2. Mengintegrasikan IPFS/Pinata sebagai penyimpanan gambar *off-chain* yang menghasilkan CID.
3. Menyimpan metadata gambar pada smart contract Ethereum.
4. Menggunakan React dan Ether.js untuk menghubungkan pengguna dengan blockchain melalui MetaMask.
5. Menyediakan fitur posting, komentar, dan likes berbasis blockchain.

1.3 Manfaat Penelitian

Penelitian ini memberikan sejumlah manfaat yang dapat dilihat dari berbagai aspek, misalnya manfaat akademis, teknis, dan praktis sebagai berikut:

1. Manfaat Akademis
   - Memberikan contoh nyata implementasi Web3.
   - Menjadi acuan pembelajaran integrasi blockchain, IPFS, React.
2. Manfaat Teknis
   - Memperlihatkan proses pengembangan aplikasi *full blockchain*.
   - Menjelaskan metode pengunggahan file ke IPFS menggunakan Pinata.
3. Manfaat Praktis
   - Menyediakan sistem media sosial tanpa kontrol terpusat.
   - Menjamin permanensi data yang tidak dapat dimodifikasi.

2. METODOLOGI

Bagian metodologi menjelaskan pendekatan yang digunakan dalam merancang dan membangun sistem.

2.1 Arsitektur Sistem

Arsitektur sistem berikut menggambarkan komponen-komponen utama yang digunakan dalam pembangunan platform beserta peran masing-masing dalam alur kerja aplikasi:

1. Frontend React, Upload gambar, menampilkan feed, mengeksekusi transaksi smart contract.
2. MetaMask, Wallet dan signer transaksi.
3. Ether.js, Penghubung antara React dan smart contract.
4. IPFS/Pinata, Penyimpanan file gambar secara *distributed*.
5. Smart Contract Ethereum, Penyimpanan metadata CID, likes, dan komentar.

2.2 Tahapan Pengembangan

Tahapan pengembangan berikut menggambarkan langkah-langkah yang ditempuh secara runtut untuk memastikan seluruh fitur dapat terimplementasi sesuai desain sistem:

1. Mendesain smart contract untuk posting, like, dan komentar.
2. Mengimplementasikan mekanisme upload gambar ke IPFS/Pinata.
3. Menghubungkan React–MetaMask menggunakan Ether.js.
4. Menyusun UI feed, komentar, profil, dan galeri.

3. IMPLEMENTASI SISTEM

Pada tahap implementasi, seluruh komponen utama yang telah dirancang pada bagian metodologi mulai diwujudkan ke dalam bentuk sistem nyata. Implementasi dilakukan dengan mengintegrasikan smart contract sebagai pusat logika, IPFS sebagai penyimpanan file terdesentralisasi, serta React–MetaMask sebagai antarmuka pengguna dan penghubung ke blockchain. Bagian berikut menjelaskan detail penerapan masing-masing komponen dalam sistem.

3.1 Implementasi Smart Contract

Smart contract dikembangkan menggunakan bahasa Solidity dan dijalankan pada jaringan Ethereum (testnet). Kontrak berperan sebagai pusat logika bisnis aplikasi karena seluruh data postingan, komentar, username, dan likes disimpan langsung di blockchain. Struktur utama yang digunakan terdiri dari:

1. Struct Post yang digunakan untuk menyimpan CID, alamat pengunggah, timestamp, jumlah like, serta daftar komentar.
2. Struct Comment yang digunakan untuk menyimpan alamat pemberi komentar, isi komentar, dan waktu komentar dibuat.

Berikut beberapa fungsi yang diimplementasikan:

1. setUsername()

   Menyimpan username setiap pengguna pada blockchain. Username terikat pada alamat wallet pengguna sehingga mudah diverifikasi dan tidak bergantung pada server eksternal.
2. createPost()

   Membuat postingan baru dengan memasukkan CID dari IPFS. Fungsi ini menyimpan metadata seperti pengunggah, waktu, dan jumlah like awal.
3. likePost()

   Memberikan like pada postingan tertentu. Kontrak menggunakan *mapping* hasLiked untuk memastikan satu akun hanya dapat melakukan like satu kali, sehingga mencegah spam like.

4. addComment()

   Menambahkan komentar ke dalam array Comment[] milik sebuah postingan. Setiap komentar langsung tersimpan pada blockchain dan bersifat permanen.

5. getPost / getComment / getCommentCount

   Berfungsi sebagai pembaca data (read-only) untuk mengambil informasi postingan dan komentar agar dapat ditampilkan di frontend.

Semua fungsi ini dijalankan melalui transaksi MetaMask sehingga seluruh interaksi bersifat terdesentralisasi. Kode lengkapnya ditampilkan pada Lampiran 1.

3.2 Implementasi Upload ke IFPS

Proses upload gambar dilakukan sepenuhnya pada sisi frontend menggunakan React. Aplikasi memanfaatkan fetch API, JWT Token Pinata, dan endpoint /pinning/pinFileToIPFS untuk mengirim file ke jaringan IPFS melalui layanan Pinata.

Kode lengkap ada pada Lampiran 2**.** Alur upload berlangsung sebagai berikut:

1. Pengguna memilih file gambar melalui form pada antarmuka React.
2. React membungkus file tersebut ke dalam format multipart/form-data.
3. File dikirim ke Pinata menggunakan permintaan HTTP yang dilengkapi JWT Token sebagai autentikasi.
4. Pinata menyimpan file pada jaringan IPFS.
5. Server Pinata mengembalikan sebuah Content Identifier (CID) yang menjadi identitas permanen gambar tersebut.
6. CID kemudian dikirim ke smart contract melalui MetaMask untuk dicatat sebagai metadata postingan.

CID yang tersimpan di blockchain tersebut digunakan kembali oleh frontend untuk menampilkan gambar melalui Pinata Gateway. Dengan pendekatan ini, gambar tidak perlu disimpan pada server atau database terpusat, sehingga penyimpanan konten menjadi lebih aman dan terdesentralisasi. Kode implementasi lengkap ditampilkan pada Lampiran 2.

3.3 Integrasi React dan Smart Contract

Integrasi antara React dan smart contract dilakukan dengan menggunakan library Ether.js. Library ini memungkinkan frontend berkomunikasi langsung dengan blockchain melalui MetaMask. Tahapan integrasi:

1. Inisialisasi Provider

   React mendeteksi MetaMask melalui window.ethereum dan membuat provider blockchain.

2. Mendapatkan Signer

   Signer digunakan untuk menandatangani transaksi seperti createPost, likePost, atau addComment.

3. Menghubungkan ABI dan Contract Address

   ABI (Application Binary Interface) dan alamat kontrak digunakan untuk membuat objek kontrak Ethereum di frontend.

4. Pemanggilan Fungsi Smart Contract

   React memanggil fungsi seperti createPost(cid) atau addComment() melalui metode Ether.js, lalu MetaMask akan meminta konfirmasi transaksi.

5. Pengambilan Data untuk UI

   Fungsi read-only seperti getPost() dipanggil untuk mengambil data postingan, komentar, jumlah like, dan menampilkannya di feed maupun galeri.

Integrasi ini membuat seluruh interaksi antara aplikasi dan blockchain berlangsung secara langsung tanpa middleware atau backend. Kode lengkap ada di Lampiran 3.

3.4 Fitur Aplikasi

Platform media sosial berbasis blockchain yang dikembangkan dalam penelitian ini memiliki beberapa fitur utama yang semuanya berjalan di atas arsitektur terdesentralisasi. Setiap fitur diuji dan berfungsi sesuai rancangan. Penjelasan lengkapnya adalah sebagai berikut:

1. Upload Gambar ke IPFS

   Pengguna dapat mengunggah gambar melalui React, dan file tersimpan di IPFS melalui Pinata. Setiap upload menghasilkan CID yang bersifat permanen dan dapat diakses melalui Pinata Gateway.

2. Posting Metadata ke Blockchain

   Setelah memperoleh CID, pengguna dapat memposting gambar ke smart contract. Metadata seperti CID, alamat pengunggah, waktu posting, jumlah like, dan komentar tersimpan secara immutable di blockchain.

3. Like Postingan

   Smart contract menyediakan fitur like dengan validasi mapping "`hasLiked`" sehingga setiap akun hanya dapat memberikan satu like pada setiap postingan.

4. Komentar Berbasis Blockchain

   Pengguna dapat memberi komentar yang kemudian disimpan sebagai struktur Comment di smart contract. Komentar bersifat permanen dan ditampilkan kembali melalui frontend.

5. Galeri Gambar

   React menampilkan seluruh postingan gambar berdasarkan CID yang tersimpan di blockchain dan ditarik dari IPFS tanpa memerlukan server penyimpanan.

6. Halaman Feed

   Sistem menyediakan feed yang menampilkan seluruh postingan dari semua pengguna. Setiap entri feed menampilkan gambar dari IPFS, jumlah like, daftar komentar, serta alamat pembuat postingan. Feed diambil langsung dari blockchain sehingga urutan dan datanya benar-benar sesuai kondisi on-chain.

7. Profil Pengguna

   Pengguna dapat mengatur dan menampilkan username melalui fungsi "`setUsername`". Username disimpan di blockchain sehingga selalu dapat diverifikasi dan tidak bergantung pada server eksternal.

4. HASIL DAN PEMBAHASAN

Hasil implementasi menunjukkan bahwa seluruh komponen utama seperti smart contract, IPFS/Pinata, React, MetaMask, dan Ether.js berhasil terhubung dan membentuk satu sistem yang dapat berjalan tanpa backend terpusat. Sistem beroperasi sepenuhnya dengan memanfaatkan smart contract sebagai pusat logika bisnis dan IPFS sebagai penyimpanan file.

Pertama, proses integrasi blockchain, IPFS, React berjalan dengan baik. Gambar yang diunggah melalui frontend berhasil diproses oleh Pinata, menghasilkan CID yang konsisten dan dapat diakses melalui gateway IPFS. CID ini kemudian diteruskan ke smart contract sebagai metadata postingan dan ditampilkan kembali pada antarmuka React. Smart contract mampu menangani fungsi inti media sosial pada prototipe ini. Pembuatan postingan berhasil menyimpan CID, alamat pengunggah, dan timestamp secara immutable. Fitur like bekerja dengan validasi melalui mapping "`hasLiked`" sehingga setiap akun hanya bisa memberikan satu like pada satu postingan. Mekanisme komentar juga berjalan baik, di mana setiap komentar tersimpan dalam array struktur Comment yang terkait dengan postingan.

Di sisi frontend, React berhasil menampilkan seluruh data on-chain secara dinamis. Feed postingan, galeri gambar, jumlah like, serta komentar dapat ditampilkan berdasarkan data yang diambil langsung dari kontrak. Interaksi pengguna seperti membuat postingan, menambahkan komentar, dan memberikan like dapat diproses melalui MetaMask dan direfleksikan kembali secara real-time.

Secara keseluruhan, hasil implementasi menunjukkan bahwa sistem dapat berfungsi tanpa server backend dan tetap mampu menyediakan fitur dasar media sosial berbasis gambar dengan memanfaatkan blockchain dan IPFS. Integrasi teknologi berjalan stabil, dan seluruh alur aplikasi dapat dijalankan sesuai rancangan.

5. KESIMPULAN

Penelitian ini berhasil membangun sebuah platform berbagi gambar berbasis blockchain yang memanfaatkan IPFS sebagai penyimpanan file terdesentralisasi dan smart contract Ethereum sebagai pencatat metadata permanen. Seluruh fitur inti seperti pembuatan postingan, pemberian like, dan penambahan komentar dapat dijalankan sepenuhnya tanpa backend terpusat, karena seluruh logika bisnis ditangani langsung oleh smart contract. Integrasi antara React, MetaMask, dan Ether.js berjalan dengan baik sehingga pengguna dapat berinteraksi langsung dengan blockchain melalui antarmuka web. Proses upload gambar ke IPFS juga berhasil menghasilkan CID yang konsisten dan dapat ditampilkan kembali di frontend. Dengan kombinasi ini, sistem mampu menyediakan mekanisme berbagi gambar yang lebih aman, transparan, dan tidak dapat dimodifikasi.

Secara keseluruhan, implementasi yang dilakukan menunjukkan bahwa pendekatan blockchain, IPFS dapat menjadi alternatif arsitektur media sosial yang memberikan kontrol penuh kepada pengguna atas data mereka dan menghilangkan ketergantungan pada server terpusat.

DAFTAR PUSTAKA

[1] G. Zyskind and O. Nathan, “Decentralizing privacy: Using blockchain to protect personal data,” *2015 IEEE Security and Privacy Workshops (SPW)*, 2015.

[2] J. Li, W. Peng and S. Yu, “Blockchain-Based Data Integrity Verification in Cloud Storage With IPFS,” *IEEE Internet of Things Journal*, vol. 8, no. 9, 2021.

LAMPIRAN

Lampiran 1 Smart Contract

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

contract PostingImage {

    mapping(address => string) public usernames;

    struct Comment {
        address author;
        string text;
        uint256 timestamp;
    }

    struct Post {
        uint256 id;
        address author;
        string imageUrl;
        string description;
        uint256 likes;
        uint256 timestamp;
        Comment[] comments;
    }

    uint256 public totalPosts = 0;
    mapping(uint256 => Post) public posts;
    mapping(uint256 => mapping(address => bool)) public hasLiked;

    event UsernameUpdated(address indexed user, string username);

    event PostCreated(
        uint256 id,
        address indexed author,
        string imageUrl,
        string description,
        uint256 timestamp
    );

    event PostLiked(uint256 id, address indexed liker, uint256 newLikes);

    event CommentAdded(
        uint256 indexed id,
        address indexed commenter,
        string text,
        uint256 timestamp
    );

    function setUsername(string memory name) public {
        usernames[msg.sender] = name;
        emit UsernameUpdated(msg.sender, name);
    }

    function createPost(string memory img, string memory desc) public {
        Post storage p = posts[totalPosts];
        p.id = totalPosts;
```

```
        p.author = msg.sender;
        p.imageUrl = img;
        p.description = desc;
        p.timestamp = block.timestamp;

        emit PostCreated(totalPosts, msg.sender, img, desc, block.timestamp);
        totalPosts++;
    }

    function likePost(uint256 postId) public {
        require(postId < totalPosts, "Post not found");
        require(!hasLiked[postId][msg.sender], "Anda sudah like postingan
ini");

        posts[postId].likes++;

        // Penting → tandai bahwa user sudah like
        hasLiked[postId][msg.sender] = true;

        emit PostLiked(postId, msg.sender, posts[postId].likes);
    }

    function addComment(uint256 postId, string memory text) public {
        require(postId < totalPosts, "Post not found");

        posts[postId].comments.push(
            Comment(msg.sender, text, block.timestamp)
        );

        emit CommentAdded(postId, msg.sender, text, block.timestamp);
    }

    function getCommentCount(uint256 postId) public view returns (uint256) {
        return posts[postId].comments.length;
    }

    function getComment(uint256 postId, uint256 index)
        public
        view
        returns (address, string memory, uint256)
    {
        Comment storage c = posts[postId].comments[index];
        return (c.author, c.text, c.timestamp);
    }

    function getPost(uint256 id)
        public
        view
        returns (
            uint256,
            address,
            string memory,
            string memory,
            uint256,
            uint256,
            uint256
        )
```

```
    {
        Post storage p = posts[id];
        return (
            p.id,
            p.author,
            p.imageUrl,
            p.description,
            p.likes,
            p.timestamp,
            p.comments.length
        );
    }
}
```

Smart contract *PostingImage* ini berfungsi sebagai logika utama dalam pengelolaan postingan, komentar, username, dan fitur *like* pada platform berbagi gambar berbasis blockchain. Kontrak ini menyimpan username setiap pengguna, metadata postingan, serta daftar komentar langsung di blockchain menggunakan struktur *Post* dan *Comment*. Setiap postingan memiliki ID, alamat pembuat, URL gambar (CID IPFS), deskripsi, jumlah *like*, waktu unggah, dan daftar komentar.

Kontrak juga menerapkan *mapping hasLiked* untuk memastikan bahwa setiap alamat hanya dapat memberikan satu *like* pada satu postingan. Fungsi *setUsername* digunakan untuk mencatat username pengguna, sementara *createPost* membuat postingan baru dengan menyimpan CID dan deskripsi. Fitur *likePost* menambah jumlah *like* jika pengguna belum pernah memberi *like* sebelumnya. Fungsi *addComment* menambahkan komentar baru ke dalam array komentar sebuah postingan.

Untuk kebutuhan frontend, beberapa fungsi pembaca seperti *getPost*, *getComment*, dan *getCommentCount* disediakan agar React dapat mengambil data postingan dan komentar secara langsung dari blockchain. Dengan struktur ini, smart contract memastikan seluruh data tersimpan secara permanen, transparan, dan tidak bergantung pada backend terpusat.

Lampiran 2 Upload ke IPFS

```
export async function uploadToIPFS(file) {
  const PINATA_URL = "https://api.pinata.cloud/pinning/pinFileToIPFS";

  const jwt = process.env.REACT_APP_PINATA_JWT;
  if (!jwt) throw new Error("PINATA JWT belum di-set di .env");

  const formData = new FormData();
  formData.append("file", file);

  formData.append(
    "pinataMetadata",
    JSON.stringify({ name: file.name })
  );

  formData.append(
    "pinataOptions",
    JSON.stringify({ cidVersion: 1 })
  );

  const res = await fetch(PINATA_URL, {
    method: "POST",
```

```
    headers: {
      Authorization: jwt
    },
    body: formData,
  });

  const data = await res.json();

  if (!data.IpfsHash) throw new Error("Pinata tidak mengembalikan CID!");

  const GATEWAY = process.env.REACT_APP_PINATA_GATEWAY;
  return `${GATEWAY}/ipfs/${data.IpfsHash}`;
}
```

Fungsi uploadToIPFS digunakan untuk mengunggah file gambar ke IPFS melalui layanan Pinata. Proses diawali dengan mendefinisikan URL endpoint Pinata dan mengambil JWT Token dari variabel lingkungan .env untuk keperluan autentikasi. File yang dipilih pengguna dimasukkan ke dalam FormData, bersama metadata seperti nama file dan opsi cidVersion. Permintaan upload kemudian dikirim menggunakan fetch dengan metode POST dan header otorisasi JWT. Setelah Pinata memproses file, server akan mengembalikan objek JSON berisi IpfsHash yang menjadi CID file. CID ini kemudian digabungkan dengan URL gateway Pinata untuk menghasilkan tautan IPFS yang dapat langsung digunakan pada aplikasi frontend. Jika proses gagal atau Pinata tidak mengembalikan CID, fungsi akan melempar error sebagai penanda kegagalan upload.

Lampiran Integrasi Contract

```
// client/src/utils/contract.js
import { ethers } from "ethers";
import ABI from "../abi/PostingImage.json";

const CONTRACT_ADDRESS = process.env.REACT_APP_CONTRACT_ADDRESS;

export const getContract = async () => {
  if (!window.ethereum) {
    throw new Error("MetaMask tidak ditemukan. Tolong install MetaMask.");
  }

  if (!CONTRACT_ADDRESS) {
    throw new Error("REACT_APP_CONTRACT_ADDRESS belum diset di file .env");
  }

  const provider = new ethers.BrowserProvider(window.ethereum);
  const signer = await provider.getSigner();

  const contract = new ethers.Contract(CONTRACT_ADDRESS, ABI.abi, signer);
  return contract;
};
```

Kode pada berkas contract.js berfungsi untuk menginisialisasi koneksi antara frontend React dan smart contract yang telah dideploy di blockchain. Fungsi getContract terlebih dahulu memastikan bahwa MetaMask tersedia pada browser serta alamat kontrak telah diset melalui variabel lingkungan .env. Selanjutnya, aplikasi membuat objek BrowserProvider dari Ether.js untuk mengakses jaringan blockchain melalui MetaMask. Dari provider tersebut diambil *signer*, yaitu identitas pengguna yang digunakan untuk menandatangani transaksi. Setelah itu, objek smart

contract dibuat menggunakan alamat kontrak, ABI hasil kompilasi, dan *signer* yang telah diperoleh. Fungsi ini akan mengembalikan objek kontrak yang siap digunakan untuk memanggil fungsi-fungsi seperti membuat postingan, memberi like, atau menambahkan komentar secara langsung dari frontend.

Lampiran 4 Frontend

```
// client/src/App.js
import React, { useState, useEffect } from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  NavLink,
  Navigate,
} from "react-router-dom";
import "./App.css";

import Home from "./pages/Home";
import Profile from "./pages/Profile";
import { getContract } from "./utils/contract";

function App() {
  const [account, setAccount] = useState(null);
  const [contract, setContract] = useState(null);
  const [loadingContract, setLoadingContract] = useState(false);

  // CONNECT WALLET
  const connectWallet = async () => {
    if (!window.ethereum)
      return alert("Install MetaMask terlebih dahulu!");

    try {
      // 1. Paksa MetaMask buka popup pilihan akun
      await window.ethereum.request({
        method: "wallet_requestPermissions",
        params: [{ eth_accounts: {} }],
      });

      // 2. Ambil akun setelah user memilih
      const accounts = await window.ethereum.request({
        method: "eth_requestAccounts",
      });

      if (accounts.length > 0) {
        setAccount(accounts[0]);
      }

    } catch (err) {
      console.error("User rejected:", err);
    }
  };

  // LOAD CONTRACT SETIAP KALI ACCOUNT BERUBAH
  useEffect(() => {
    const init = async () => {
      if (!account) {
```

```
        setContract(null);
        return;
      }
      try {
        setLoadingContract(true);
        const c = await getContract();
        setContract(c);
      } catch (err) {
        console.error("Gagal load contract:", err);
        alert("Gagal menghubungkan ke smart contract. Cek console.");
      } finally {
        setLoadingContract(false);
      }
    };
    init();
  }, [account]);

  // LISTENER AKUN / NETWORK
  useEffect(() => {
    if (!window.ethereum) return;

    const handleAccountsChanged = (accounts) => {
      if (accounts.length === 0) {
        setAccount(null);
        setContract(null);
      } else {
        setAccount(accounts[0]);
      }
    };

    const handleChainChanged = () => {
      window.location.reload();
    };

    window.ethereum.on("accountsChanged", handleAccountsChanged);
    window.ethereum.on("chainChanged", handleChainChanged);

    return () => {
      window.ethereum.removeListener("accountsChanged",
handleAccountsChanged);
      window.ethereum.removeListener("chainChanged", handleChainChanged);
    };
  }, []);

  const logoutWallet = () => {
    setAccount(null);
    setContract(null);
    setTimeout(() => window.location.reload(), 150);
  };

  return (
    <Router>
      <div className="App">
        {/* NAVBAR MODIFIED */}
        <nav className="navbar">
          <div className="navbar-inner">
            <h2 className="navbar-logo">MetaSnap</h2>
```

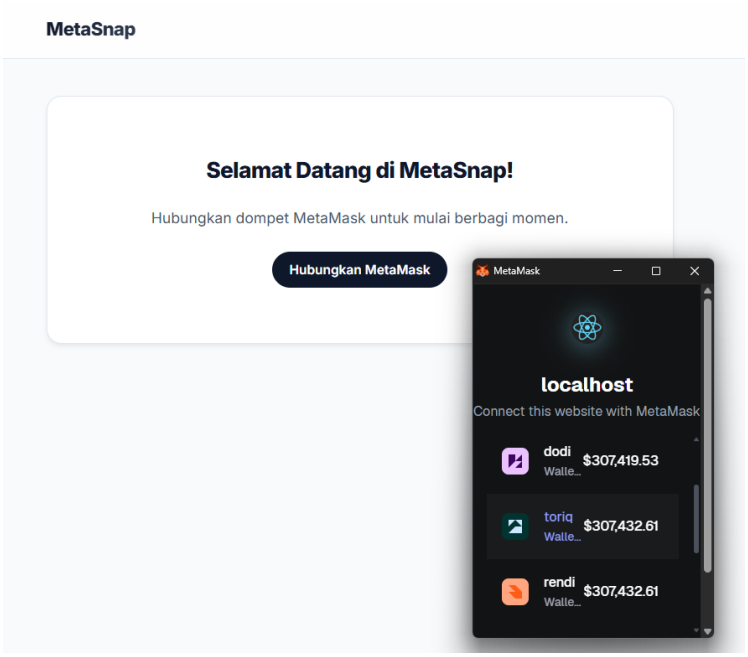
```
            <div className="nav-links">
              {account ? (
                <>
                  <NavLink
                    to="/"
                    className={({ isActive }) => (isActive ? "nav-item
active" : "nav-item")}
                  >
                    Feed
                  </NavLink>
                  <NavLink
                    to="/profile"
                    className={({ isActive }) => (isActive ? "nav-item
active" : "nav-item")}
                  >
                    Profil
                  </NavLink>

                  <span className="badge-address">
                    {account.substring(0, 6)}...
                    {account.substring(account.length - 4)}
                  </span>

                  <button          className="btn-secondary          btn-sm"
onClick={logoutWallet}>
                    Logout
                  </button>
                </>
              ) : null /* Tombol Connect dihapus di sini sesuai permintaan
*/}
            </div>
          </div>
        </nav>

        {/* KONTEN */}
        <div className="container">
          <Routes>
            <Route
              path="/"
              element={
                account ? (
                  <Home
                    account={account}
                    contract={contract}
                    loadingContract={loadingContract}
                  />
                ) : (
                  <LoginPage connectWallet={connectWallet} />
                )
              }
            />
            <Route
              path="/profile"
              element={
                account ? (
                  <Profile
```

```
                    account={account}
                    contract={contract}
                    loadingContract={loadingContract}
                  />
                ) : (
                  <Navigate to="/" />
                )
              }
            />
          </Routes>
        </div>
      </div>
    </Router>
  );
}

function LoginPage({ connectWallet }) {
  return (
    <div  className="card"  style={{  textAlign:  "center",  padding:  "60px
20px" }}>
      <h2>Selamat Datang di MetaSnap!</h2>
      <p style={{marginBottom: "24px"}}>Hubungkan dompet MetaMask untuk mulai
berbagi momen.</p>
      <button className="btn-primary" onClick={connectWallet}>
        Hubungkan MetaMask
      </button>
    </div>
  );
}

export default App;
```

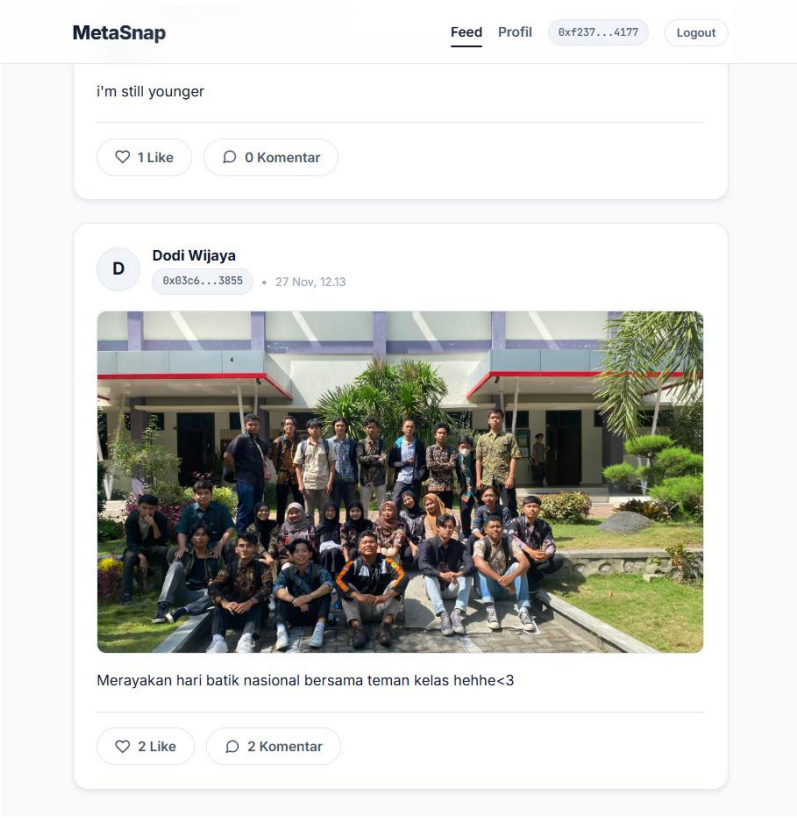
Kode pada berkas App.js berfungsi sebagai komponen utama aplikasi React yang mengatur alur navigasi, autentikasi MetaMask, serta pengambilan objek smart contract. Aplikasi menggunakan *React Router* untuk memisahkan halaman Feed dan Profile, dan menampilkan konten yang berbeda berdasarkan status wallet pengguna. Fungsi connectWallet digunakan untuk meminta izin akses akun dari MetaMask dan menyimpan alamat wallet ke dalam state. Ketika alamat wallet tersimpan, *useEffect* akan memanggil getContract untuk membuat objek kontrak sehingga frontend dapat berinteraksi dengan smart contract.

Selain itu, aplikasi memasang *listener* untuk mendeteksi perubahan akun atau jaringan pada MetaMask. Jika pengguna mengganti akun, antarmuka akan menyesuaikan secara otomatis, dan jika jaringan berubah, halaman akan dimuat ulang untuk menjaga sinkronisasi dengan blockchain. Navbar menampilkan tautan navigasi serta alamat wallet pengguna dalam bentuk singkat. Bila pengguna belum terhubung, halaman Feed menampilkan komponen *LoginPage* yang berisi tombol untuk menghubungkan MetaMask. Dengan struktur ini, komponen App.js berperan sebagai pengatur utama alur interaksi pengguna, autentikasi wallet, dan integrasi kontrak dalam aplikasi React.

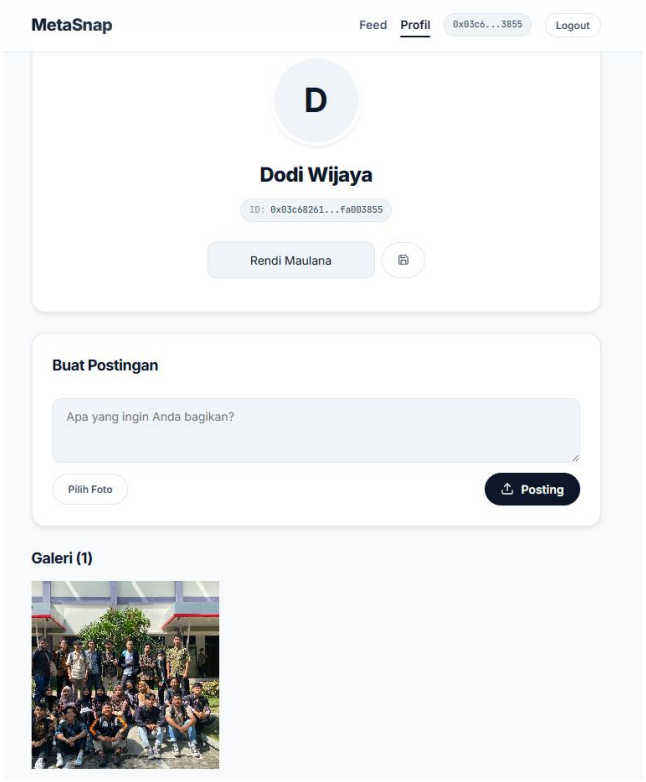
Lampiran 5 UI Aplikasi



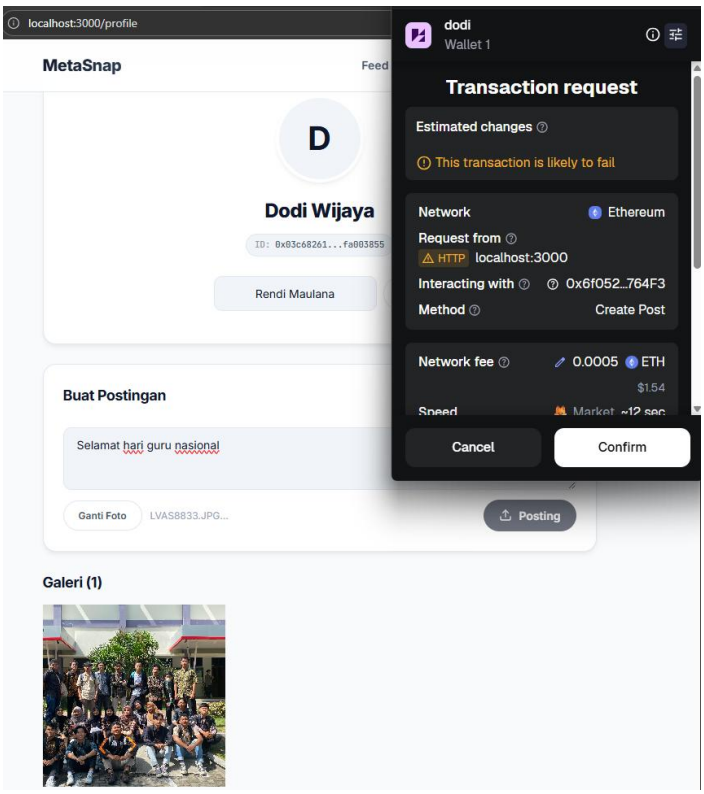
Gambar 1 Menghubungkan dengan Akun Metamask



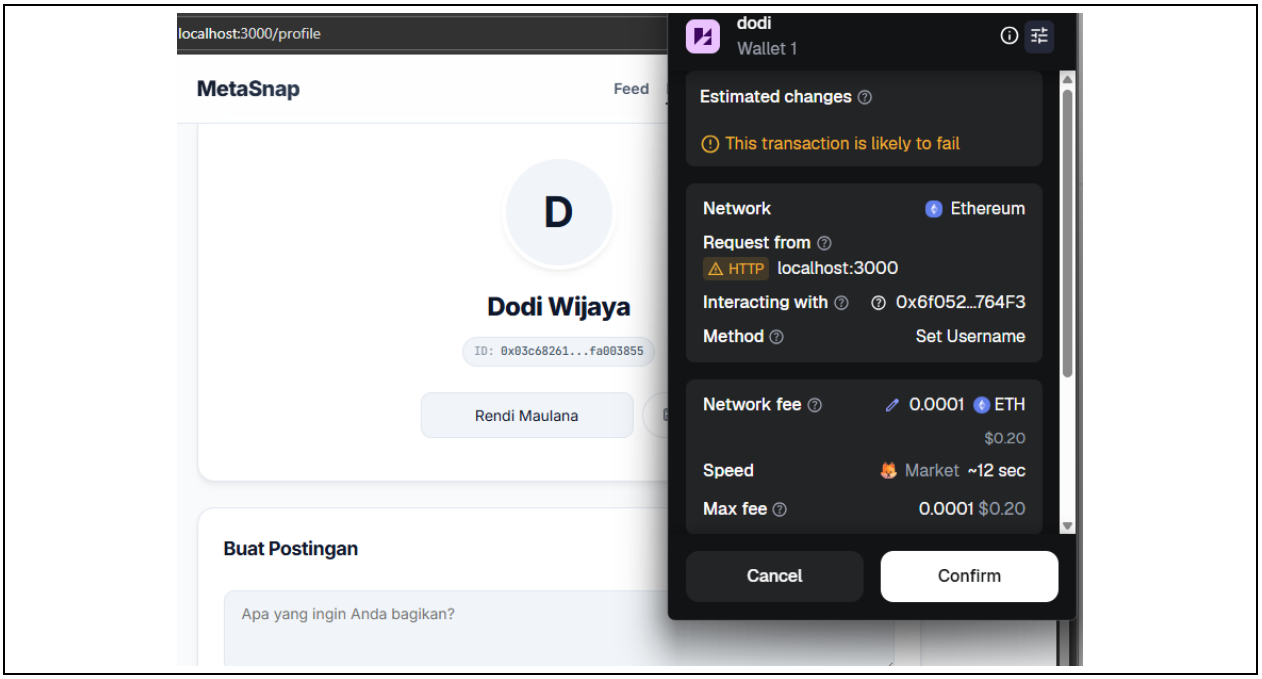
Gambar 2 Tampilan Home/Beranda



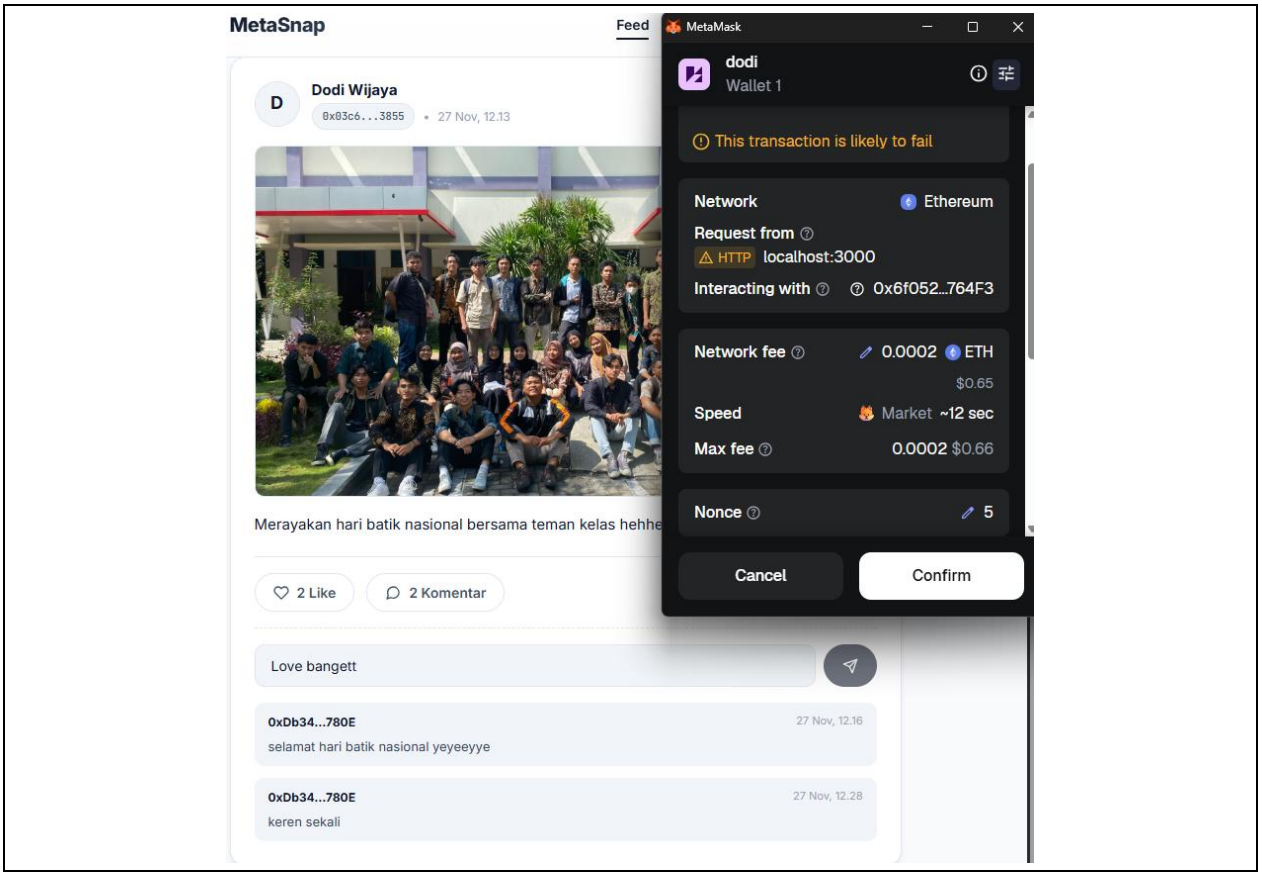
Gambar 3 Tampilan Profile



Gambar 4 Tampilan Membuat Postingan



Gambar 5 Tampilan Update Username



Gambar 6 Tampilan Ketika Memberikan Komentar