Project Report
IF3130 - Blockchain

Property Administration dApp with Ethereum

Disusun oleh :

13521050 Naufal Syifa Firdaus
13521079 Hobert Anthony Jonatan
13521118 Ahmad Ghulam Ilham

SEKOLAH TINGGI ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Problem Statement

Administrasi properti secara tradisional mempunyai banyak kelemahan dan tantangan yang dapat diselesaikan dengan aplikasi terdesentralisasi. Berikut merupakan masalah utama yang akan diatasi dengan aplikasi terdesentralisasi.

a. **Valuasi properti yang tidak terpercaya**

   Properti seringkali dihargai oleh pemilik atau broker untuk mengambil keuntungan yang sebanyak-banyaknya. Dengan dApp, valuasi properti dan riwayatnya jelas tercatat sehingga pembeli dapat dengan mudah mengetahui harga aslinya untuk pengambilan keputusan.

b. **Biaya transaksi yang tinggi**

   Pada transaksi properti seringkali digunakan broker, agen, dan pihak ketiga lainnya yang mengambil keuntungan dan meningkatkan biaya. DApp mengeliminasi hal tersebut sehingga biaya tambahan yang diperhitungkan hanya pajak saja.

c. **Sengketa kepemilikan**

   Riwayat kepemilikan tradisional seringkali tidak akurat dan ambigu. Tidak sedikit properti di Indonesia yang terkena sengketa kepemilikan karenanya. Riwayat yang jelas dan permanen diperlukan untuk menghindari sengketa.

d. **Transparansi pada riwayat kepemilikan**

   Mengacu pada poin sebelumnya, pada *property listing* tradisional, tidak sedikit informasi tentang kepemilikan yang tidak transparan. Riwayat yang permanen dapat mencegah penjualan yang curang, kepemilikan palsu, dan pemalsuan harga.

Use Case

Aplikasi dApps yang dibangun difokuskan untuk mengatasi masalah yang telah dideskripsikan sebelumnya. Fungsi utama dari aplikasi adalah sebagai platform daftar dan administrasi properti. Terdapat dua fungsionalitas utama sebagai berikut.

a. Pendaftaran dan pembaruan data properti (create, read, update)
b. Pembelian properti (transfer kepemilikan)

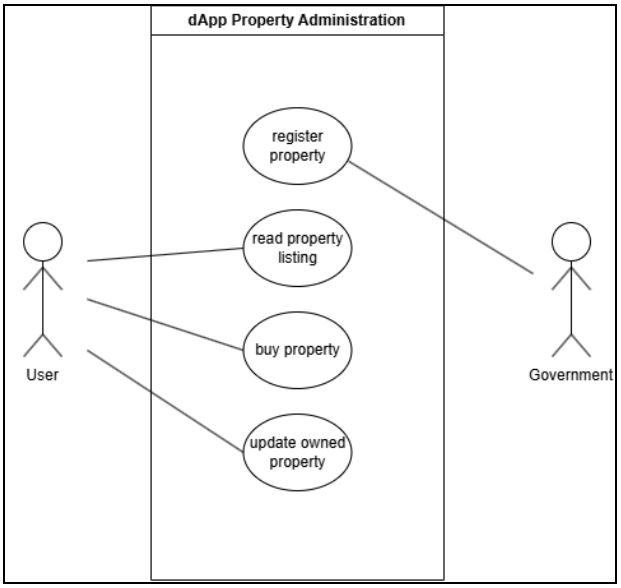


Diagram use case diatas mengilustrasikan interaksi antara user, government, dan aplikasi dApps. Otoritas government terlibat untuk memverifikasi dan menyetujui registrasi property untuk memastikan prosedurnya sesuai dengan regulasi dan hukum yang berlaku. Aplikasi perlu sistem pendukung pada government untuk verifikasi dan legalitas properti itu sendiri. User sebagai pengguna dapat melihat daftar properti yang terdapat dalam blockchain dan melakukan pembelian melalui teknologi blockchain yang transparan dan terpercaya. User juga dapat memperbarui properti yang dipunyai. Pendekatan ini mengurangi keterlibatan pihak ketiga, meningkatkan keamanan, dan meningkatkan efisiensi.

Platform Blockchain yang digunakan

Platform blockchain yang dipilih adalah Geth (go-ethereum) dengan tool Kurtosis untuk menjalankan containerized ethereum-package pada Docker. Geth adalah implementasi Ethereum dalam bahasa Go yang mendukung pengembangan blockchain pada private network. Mekanisme konsensus Geth adalah PoS (Proof-of-Stake). Justifikasi pemilihan platform Geth sebagai berikut.

1. Geth lebih cocok dengan use case yang diinginkan, yaitu sebuah dApp B2C (Business-to-Consumer). Transaksi pada blockchain berkaitan erat dengan bidang keuangan, yaitu jual beli antara pemilik properti.
2. Geth mendukung integrasi dengan dengan wallet Metamask yang sudah pernah dipelajari sehingga lebih familier dalam implementasi dan integrasi dengan wallet.
3. Geth memiliki ekosistem, tools, library, dan framework yang luas dan terdokumentasi dengan baik sehingga memudahkan pengembangan smart contract.
4. Geth memiliki implementasi token ERC-721 yang dapat langsung digunakan sehingga tidak perlu mengimplementasikan token sendiri.

5. Geth memiliki standar dan fitur keamanan yang baik sehingga dapat meminimalisasi risiko keamanan pada smart contract.

Tech Stack

Frontend

1. Next.js : Next sebagai framework web development utama dipilih berdasarkan simplisitasnya dan integrasinya dengan library blockchain yang akan digunakan. Beberapa keuntungannya seperti.

    - Fitur API Routes untuk komunikasi langsung dengan smart contract dan nodes tanpa backend terpisah
    - Integrasi dengan ethers.js dan metamask yang baik

2. Ethers.js : Library JavaScript untuk interaksi antara frontend dengan smart contract. Berikut justifikasi singkat penggunaan Ethers.js.

    - Lebih modern, ringan, modular, dan mudah digunakan dibandingkan Web3.js
    - Dokumentasi yang lengkap untuk keperluan pengembangan dan debugging

Wallet

1. MetaMask : Wallet cryptocurrency berbasis browser untuk pengelolaan aset digital dan melakukan transaksi pada Ethereum. Berikut justifikasi singkat penggunaan MetaMask.

    - Merupakan wallet populer yang memenuhi standar industri pada dApps berbasis Ethereum
    - Keamanan yang terjamin berdasarkan mekanisme pengelolaan private key dan network sehingga mendukung penggunaan private network

Smart contract

1. Solidity : Bahasa pemrograman untuk pengembangan smart contract pada Ethereum Virtual Machine (EVM). Berikut justifikasi singkat penggunaan Solidity.

    - Community support yang luas sehingga banyak referensi, tutorial, tools, dan dokumentasi yang tersedia
    - Kemudahan integrasi dengan framework Hardhat dan OpenZeppelin

2. Hardhat : Framework untuk pengembangan, pengujian, dan deployment smart contract pada Ethereum. Berikut justifikasi singkat penggunaan Hardhat.

    - Setup smart contract dan konfigurasi network account sederhana
    - Kemudahan dalam melakukan compile dan deploy smart contract

3. OpenZeppelin : Library contract Solidity yang menyediakan berbagai fitur tambahan untuk pengembangan smart contract. Berikut justifikasi penggunaan OpenZeppelin.

    - Menyediakan implementasi ERC-721 sebagai tokenisasi properti yang memenuhi standar industri
    - Menyediakan implementasi Ownable dan ReentrancyGuard sebagai mekanisme keamanan tambahan

Oracle

1. Solidity : Bahasa pemrograman untuk pengembangan smart contract untuk implementasi on-chain oracle atau oracle yang berada dalam lingkup blockchain. Justifikasi penggunaannya karena mengikuti implementasi yang digunakan pada smart contract.

External API

1. Node.js : Runtime environment berbasis JavaScript untuk menjalankan kode JavaScript di luar browser. Node.js digunakan untuk membangun aplikasi server-side menggunakan javascript atau typescript. Berikut justifikasi singkat penggunaan Node.js dalam tugas ini.

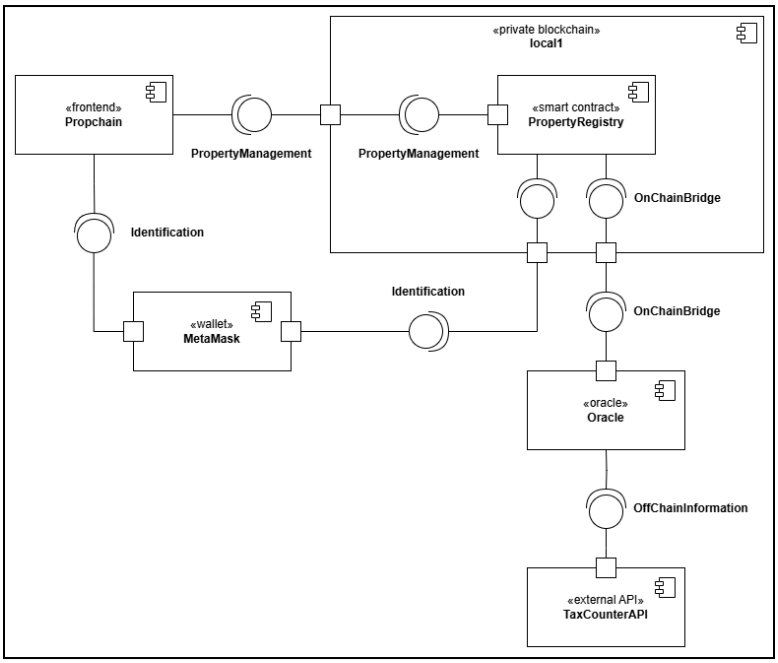
    - Familiaritas dengan javascript yang sudah biasa digunakan untuk pengembangan frontend, dapat digunakan juga untuk pengembangan backend dengan menggunakan Node.js.

    - Node.js dirancang untuk aplikasi ringan yang membutuhkan kecepatan respons tinggi. Tidak memerlukan banyak sumber daya untuk server sederhana.

2. Express.js : Framework untuk Node.js yang secara signifikan menyederhanakan pengembangan aplikasi server-side. Berikut justifikasi singkat penggunaan Express.js.

    - Express mengurangi *boilerplate code* dan menyediakan antarmuka yang lebih *clean* untuk membuat endpoint API sehingga dapat mempercepat proses pengembangan. Salah satu contohnya adalah pendefinisian *routing* yang jauh lebih sederhana dibanding menggunakan Node.js saja.

High Level Design



Berdasarkan high level design, berikut komponen pada dApps.

1. Frontend (Next.js) : Merupakan antarmuka bagi pengguna untuk mengakses dan berinteraksi dengan aplikasi. Frontend berinteraksi dengan smart contract melalui Ethers.js dan terintegrasi dengan wallet MetaMask.
2. Wallet (MetaMask) : Menyediakan mekanisme autentikasi pengguna berbasis alamat wallet dan mekanisme signing transaksi yang dilakukan pada aplikasi.
3. Blockchain (Geth & Kurtosis) : Merupakan private network untuk menyimpan transaksi dan eksekusi smart contract.
4. Smart contract (Solidity) : Merupakan logika bisnis dari aplikasi dengan fungsi pendaftaran, pembaruan status jual, dan transfer kepemilikan properti.
5. Oracle (Chainlink) : Mengambil data eksternal yang relevan terhadap smart contract untuk menjaga integritas transaksi smart contract.
6. External API (Node.js) : Menyediakan data yang diperlukan untuk mendukung proses pengambilan keputusan, validasi, dan verifikasi.

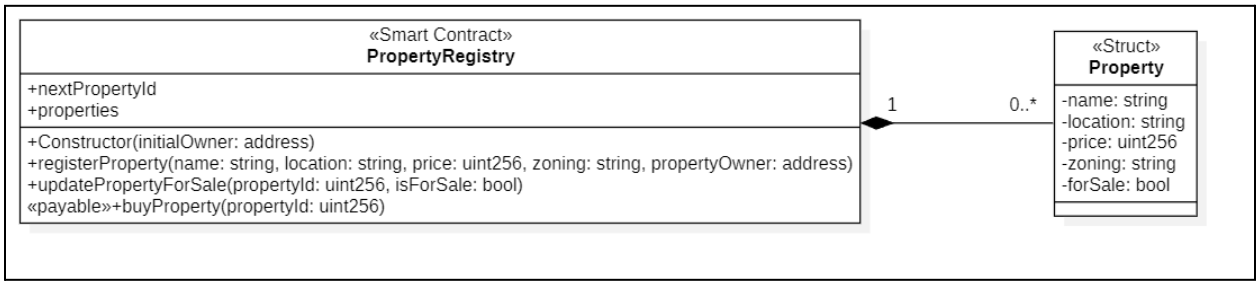
Secara umum, berikut merupakan alur interaksi antarkomponen pada dApps.

1. Sebelum melakukan transaksi, Pengguna melakukan autentikasi MetaMask menggunakan wallet address yang dimiliki. Dana untuk keperluan transaksi pengguna disimpan pada akun yang didaftarkan pada MetaMask.
2. Pengguna mengakses frontend untuk melakukan salah satu dari ketiga transaksi, yaitu pendaftaran properti, pembaruan properti, atau pembelian properti. Frontend

mengirimkan data masukan pengguna ke smart contract menggunakan Ether.js dengan signing transaksi berdasarkan wallet address MetaMask.

3. Smart contract memproses transaksi yang diteruskan oleh Ethers.js dari frontend. Ketika pemrosesan transaksi selesai, smart contract mengeluarkan event yang akan diterima oleh frontend sehingga pengguna dapat mengetahui apakah transaksi berhasil atau gagal.
4. Jika transaksi membutuhkan transfer dana, smart contract akan memproses transfer dana antara wallet address pemilik properti dan wallet address pengguna.
5. Jika transaksi membutuhkan data eksternal, smart contract akan memanggil oracle untuk meminta data eksternal yang dapat membantu proses pengambilan keputusan, validasi, dan verifikasi.
6. Oracle melakukan offchain information gathering dengan memanggil API eksternal dan menentukan apakah informasi eksternal yang didapat benar atau tidak. Jika informasi eksternal benar, smart contract akan dieksekusi.

Properti dan Method Smart Contract



Properti smart contract

1. nextPropertyId : Counter untuk menyimpan ID properti yang didaftarkan berikutnya
    - Tipe : uint256
    - Visibilitas : public
2. properties : Daftar properti menggunakan propertyId sebagai indeks
    - Tipe : mapping(uint256 => Property)
    - Visibilitas : public
    - Struct Property : Detail properti yang didaftarkan
        - name (string) : Nama properti
        - location (string) : Lokasi properti
        - price (uint256) : Harga properti
        - zoning (string) : Klasifikasi atau tipe zonasi properti
        - forSale (bool) : Status penanda apakah properti dijual

Method smart contract

1. Constructor : Menginisialisasi contract dengan nama token "PropertyToken" (simbol "PT") dan menetapkan pemilik contract

    - Parameter
        - initialOwner (address): Alamat pemilik awal contract

2. registerProperty : Mendaftarkan properti baru ke dalam smart contract dan mentransfer token ERC721 kepada pemilik properti

    - Parameter
        - name (string) : Nama properti
        - location (string): Lokasi properti
        - price (uint256): Harga properti
        - zoning (string): Zonasi properti
        - propertyOwner (address): Alamat pemilik properti
    - Modifier
        - onlyOwner: Hanya pemilik contract yang dapat memanggil
    - Event
        - Emit PropertyRegistered dengan informasi properti yang terdaftar

3. updatePropertyForSale : Mengubah status properti menjadi dijual atau tidak dijual

    - Parameter
        - propertyId (uint256) : ID properti
        - isForSale (bool) : Status dijual
    - Modifier
        - Pemanggil harus pemilik properti
    - Event
        - Emit PropertyUpdated dengan informasi status yang diperbarui

4. buyProperty : Membeli properti yang dijual dengan mengirimkan pembayaran

    - Parameter
        - propertyId (uint256) : ID properti yang akan dibeli
    - Modifier
        - nonReentrant : Mencegah serangan reentrancy
        - Properti harus dalam status dijual
    - Event
        - Emit PropertyTransferred dengan informasi perpindahan kepemilikan

Oracle

Deskripsi dan Struktur Data

Data dari API eksternal yang diambil menggunakan oracle adalah data pajak PBB (Pajak Bumi dan Bangunan) dan data pajak PPN (Pajak Pertambahan Nilai) untuk proses jual beli properti. Data diambil dari API eksternal yang diimplementasikan oleh kelompok kami sendiri untuk menyederhanakan permasalahan karena sistem yang dibuat hanya sebagai *proof of concept*. Struktur dari data yang diambil dan dikirim adalah sebagai berikut.

```
{

  building_area: Number;

  land_area: Number;

  building_price_per_m2: Number,

  land_price_per_m2: Number

}
```

Data dikirim dalam format JSON dengan empat atribut, yaitu luas bangunan (building_area), luas tanah (land_area), harga bangunan per meter persegi (building_price_per_m2), dan harga tanah per meter persegi (land_price_per_m2). Berikutnya data yang diterima sebagai hasil proses dari API adalah sebagai berikut.

```
{

  harga_akhir: Number

}
```

Data yang diterima sebagai *response* adalah harga pajak yang harus dibayar, baik itu untuk pajak PBB maupun pajak PPN.

Mekanisme Pengambilan Data dan Integrasi Smart Contract

Implementasi sistem oracle memiliki dua komponen utama, yaitu smart contract dalam blockchain dan node oracle di luar blockchain atau off-chain.

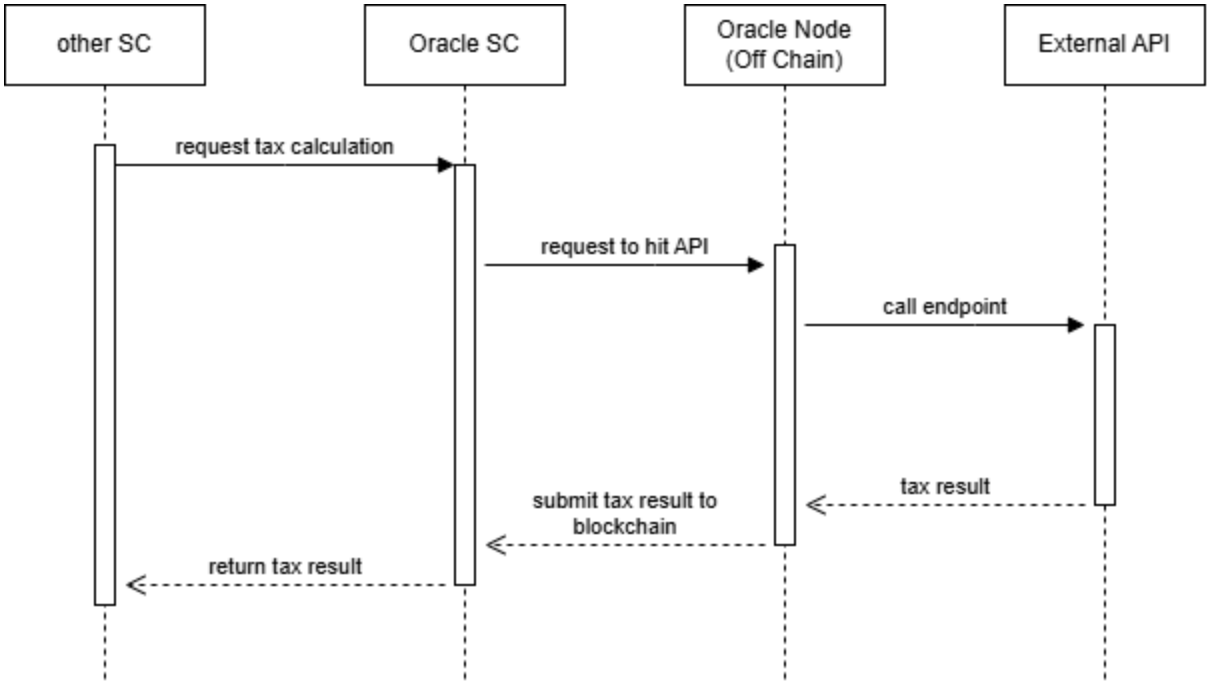
Mekanisme kerja dari node oracle (off-chain) adalah sebagai berikut:

1. Mendengarkan Event
   Node off-chain terhubung ke smart contract dengan library ethers.js dan mendengarkan event RequestCreated. Event ini memberikan detail dari request yang dikirim, meliputi requestId, tipe request, parameter perhitungan pajak, dan informasi pendukung lainnya
2. Memproses Request
   Setelah menerima event request, node kemudian menentukan jenis request yang diterimanya berdasarkan requestType yang ada pada request tersebut. Kemudian memanggil handler *function* sesuai dengan request yang didapat untuk memproses permintaan tersebut.
3. Pemanggilan API
   Dalam memproses request, dilakukan pemanggilan ke endpoint API sesuai dengan perhitungan pajak yang terdapat pada request. Hasil perhitungan pajak kemudian didapat dari endpoint API tersebut.
4. Mengirimkan Hasil ke Blockchain
   Setelah mendapatkan hasil pajak dari API, node kemudian memanggil fungsi fulfillRequest pada smart contract, mengirimkan hasil pajaknya dan menandai bahwa permintaan tersebut telah dipenuhi.

Mekanisme kerja dari Oracle pada smart contract (on-chain) adalah sebagai berikut.

1. Pembuatan Request
   Terdapat fungsi requestPBBCalculation dan requestPPNCalculation yang memungkinakan pengguna untuk membuat permintaan perhitungan nilai pajak. Sebuah requestId yang unik dihasilkan menggunakan fungsi hash keccak256. Struct OracleRequest menyimpan detail dari request yang dibuat, seperti alamat pengirim, jenis pajak, dll. Terdapat suatu event RequestCreated untuk menandakan bahwa request telah dibuat.
2. Pemenuhan Request
   Fungsi fulfillRequest akan dipanggil oleh node oracle off-chain untuk memberikan hasil perhitungan pajak yang didapat dari API eksternal. Pemenuhan request juga sekaligus ditandai dengan event DataFulfilled yang akan di emit dalam fungsi fulfillRequest.

Berikut adalah penggambaran sistem saat suatu smart contract ingin menggunakan oracle untuk mengambil data dari API eksternal.



Skenario Pengambilan Data Gagal

Saat terjadi kegagalan untuk pengambilan data, implementasi yang dibuat adalah ditetapkan suatu nilai timeout selama 30 detik untuk mengindikasikan apakah terjadi kegagalan atau tidak saat memanggil oracle.

Design Pattern

Design pattern yang digunakan dalam implementasi smart contract sebagai berikut.

1. Check-Effects-Interaction Pattern : Mencegah serangan reentrancy dengan memastikan bahwa transaksi tertentu terdiri atas 3 tahap berikut.
    - Validasi (check) : Memverifikasi apakah syarat dan kondisi transaksi terpenuhi
    - Pembaruan (effect) : Mengubah state variable hanya dilakukan setelah syarat dan kondisi transaksi terpenuhi
    - Interaksi (interact) : Interaksi dengan komponen eksternal dilakukan di akhir transaksi, contohnya melakukan transfer dana menggunakan payable
2. Access Control Pattern : Membatasi akses terhadap transaksi tertentu menggunakan mekanisme otorisasi sehingga dapat meningkatkan 3 hal berikut.
    - Keamanan : Hanya pemilik contract yang dapat mengakses transaksi tertentu
    - Integritas : Menghindari perubahan atau penyalahgunaan transaksi tertentu oleh pihak unauthorized

- Kesederhanaan : Penggunaan Ownable pada library OpenZeppelin Mengurangi risiko implementasi access control yang salah

3. Event Driven Pattern : Menggunakan event sebagai trigger atau notifikasi terhadap perubahan pada smart contract sehingga dapat meningkatkan 2 hal berikut.

    - Transparansi : Perubahan pada smart contract dicatat dan di-broadcast kepada elemen eksternal
    - Efisiensi : Memudahkan frontend mendeteksi perubahan tanpa perlu melakukan polling state

Optimasi (Optional)

Optimasi yang dilakukan pada smart contract beserta justifikasinya sebagai berikut.

1. Biaya : Mengurangi konsumsi gas untuk transaksi dan biaya deployment contract

    - Menggunakan mapping untuk menyimpan properti dibandingkan menggunakan iterasi array
    - Menggunakan fitur inheritance OpenZeppelin (ERC721, Ownable, ReentrancyGuard) untuk mengurangi kompleksitas kode

2. Keamanan : Mencegah serangan reentrancy dan unauthorized access terhadap transaksi tertentu

    - Menambahkan modifier nonReentrant pada transaksi buyProperty
    - Menambahkan modifier onlyOwner pada transaksi registerProperty

3. Skalabilitas : Memastikan contract dapat menangani data dan transaksi besar

    - Menggunakan mapping untuk menyimpan properti dibandingkan menggunakan iterasi array
    - Menggunakan event untuk mencatat dan memberi trigger atau notifikasi terhadap registrasi, pembaruan, dan transfer properti

Daftar Anggota Kelompok

NIM	**Nama**	**Pembagian Tugas**
13521050	Naufal Syifa Firdaus	Frontend, documentation
13521079	Hobert Anthony Jonatan	Oracle, external API, documentation
13521118	Ahmad Ghulam Ilham	Smart contract, integration, documentation

Link Video Demo

https://youtu.be/wkv8Jk0ty84

Referensi

- Hyperledger vs Ethereum in Blockchain - GeeksforGeeks
- Blockchain in Real Estate: Step-by-Step Guide
- Real Estate Tokenization: A Complete Guide
- Ethereum vs Fabric vs Corda: Blockchain Protocols Compared
- Design Patterns for Smart Contracts | Infuy
- Smart Contract Design Patterns in Solidity Explained! - Metana
- What are the most effective design patterns for smart contracts in FinTech?