Project Report

IF4035 - Blockchain

Oleh:

Salomo Reinhart Gregory Manalu 13521063

Michael Leon Putra Widhi 13521108

Nathan Tenka 13521172

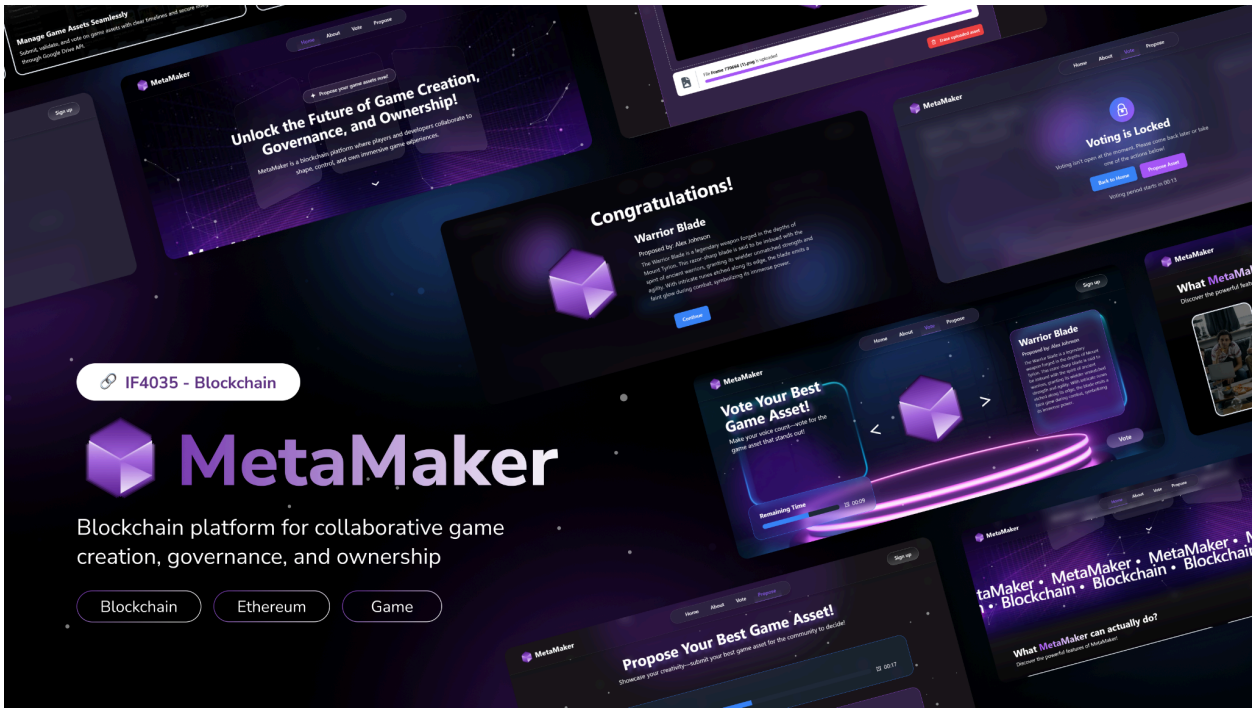
PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TINGGI ELEKTRONIK DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Problem Statement dan *Use Case*



Menyempurnakan Pengembangan Aplikasi Kolaboratif dengan Blockchain

Pengembangan aplikasi secara kolaboratif semakin mudah diakses, terutama dengan munculnya platform *open-source*. Platform ini memungkinkan siapa saja untuk memodifikasi aplikasi agar sesuai dengan kebutuhan spesifik. Jika sebuah modifikasi terbukti bernilai bagi pengguna dalam skala besar, pengembang asli dapat mengintegrasikannya ke dalam versi resmi aplikasi *open-source* tersebut.

Namun, model *open-source* ini masih menghadapi beberapa tantangan:

1. **Proses Seleksi**: Menentukan modifikasi mana yang dimasukkan ke dalam *codebase* resmi seringkali subjektif dan memakan waktu.
2. **Pelacakan Komponen**: Memastikan transparansi dan keterlacakan komponen *open-source* dalam *codebase*.
3. **Automasi Alur Kerja**: Mempercepat proses integrasi untuk meningkatkan efisiensi.

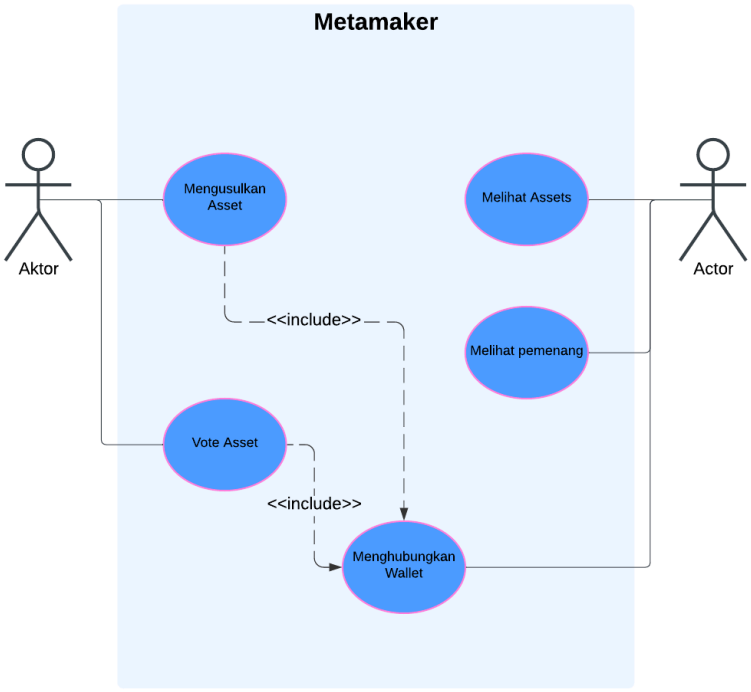
Untuk mengatasi tantangan ini, kami mengusulkan metode pengembangan publik sepenuhnya berbasis blockchain. Metode ini memungkinkan siapa saja untuk mengajukan modifikasi program dengan mengunggahnya ke blockchain dalam bentuk file yang terhubung (misalnya, link *Google Drive*). Proses penilaian dilakukan melalui voting publik, di mana modifikasi dengan suara terbanyak secara otomatis diintegrasikan ke dalam *codebase* utama setelah periode voting yang ditentukan.

Sistem berbasis blockchain ini menawarkan:

1. Keputusan yang transparan melalui voting publik.
2. Integrasi otomatis untuk modifikasi dengan suara terbanyak.
3. Peningkatan keterlacakan dan akuntabilitas untuk perubahan yang diajukan.

Metode ini dirancang untuk pengembangan gim berbasis blockchain, dengan tujuan merevolusi cara gim dikembangkan, diatur, dan dimiliki secara kolaboratif oleh komunitas. Metode ini diimplementasikan menjadi sebuah aplikasi Web3 bernama **MetaMaker**.

Berikut *use case* untuk *decentralized app* (dApp) ini :



Gambar 1. *Use Case Diagram* (UCD) MetaMaker.

Platform *Blockchain*

Aplikasi MetaMaker dikembangkan menggunakan platform Ethereum, yang dikenal sebagai salah satu platform terdepan untuk pengembangan aplikasi terdesentralisasi (*decentralized application* atau dApp).

Ada beberapa alasan utama mengapa Ethereum dipilih sebagai platform pengembangan:

1. **Dirancang untuk dApp**: Ethereum dirancang khusus untuk mendukung pengembangan dan eksekusi dApps melalui *smart contract*, memungkinkan sistem yang aman, transparan, dan bebas dari kontrol sentralisasi.
2. **Kemudahan Penggunaan**: Ethereum memiliki dokumentasi yang lengkap dan alat pengembangan yang intuitif, seperti Truffle yang dipilih untuk mengembangkan MetaMaker.
3. **Ekosistem yang Kuat**: Ethereum didukung oleh komunitas pengembang yang besar dan aktif, menyediakan berbagai sumber daya, tutorial, dan pustaka yang membantu pengembang mengatasi berbagai tantangan teknis.
4. **Keberlanjutan dan Inovasi**: Dengan transisi Ethereum ke Ethereum 2.0 (Proof of Stake), platform ini menawarkan solusi yang lebih ramah lingkungan, efisien, dan berkelanjutan untuk masa depan blockchain.

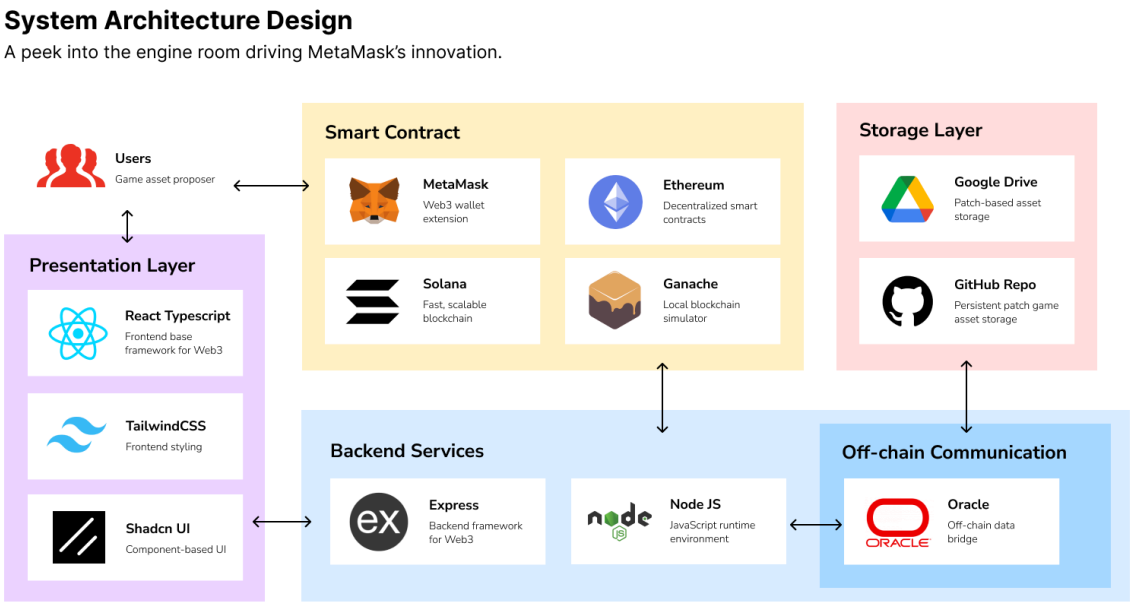
Techstack

Techstack yang digunakan sebagai berikut :

1. ***Frontend***
    - **React (TypeScript)**: Digunakan sebagai kerangka kerja utama untuk mengembangkan *frontend* berbasis Web3 karena dukungannya yang kuat terhadap integrasi dengan wallet MetaMask serta interaksi dengan blockchain.
    - **TailwindCSS**: Diterapkan untuk *styling* yang efisien dan dapat disesuaikan, memastikan antarmuka pengguna yang bersih dan responsif.
    - **Shadcn UI**: Dimanfaatkan untuk membangun UI berbasis komponen, meningkatkan modularitas dan penggunaan ulang kode.

2. ***Backend* dan Oracle**
    - **Node.js**: Dipilih sebagai *runtime environment* untuk *backend* karena kompatibilitasnya yang baik dengan *library* blockchain (contohnya Web3.js) dan ekosistem JavaScript.
    - **Express.js**: Digunakan sebagai *framework backend* untuk menangani API serta komunikasi dengan layanan *off-chain* dan jaringan blockchain.
    - **Oracle**: Berfungsi sebagai jembatan antara data *off-chain* (contohnya Google Drive) dan data *on-chain*, memfasilitasi interaksi seperti pengajuan aset dan voting.
3. ***Smart Contract***
    - **Ethereum**: Dipilih sebagai platform blockchain untuk mendesain dan mengimplementasikan *smart contract* terdesentralisasi, dengan ekosistem yang matang dan dukungan komunitas yang luas.
    - **Solana**: Digunakan untuk skalabilitas dan kemampuan transaksi cepat, meningkatkan performa sistem secara keseluruhan.
    - **Ganache**: Dimanfaatkan untuk simulasi blockchain lokal, memungkinkan pengujian dan pengembangan *smart contract* secara cepat.
    - **Truffle**: Digunakan untuk mengelola kompilasi, *deployment*, dan pengujian *smart contract* dengan lebih mudah.
4. ***Storage Layer***:
    - **Google Drive**: Digunakan untuk penyimpanan sementara aset gim yang diajukan dalam bentuk *patch*.
    - **GitHub Repository**: Berfungsi sebagai repositori utama untuk menyimpan *codebase* final dan aset yang telah disetujui.

High Level Architecture Design



Gambar 2. Arsitektur Sistem.

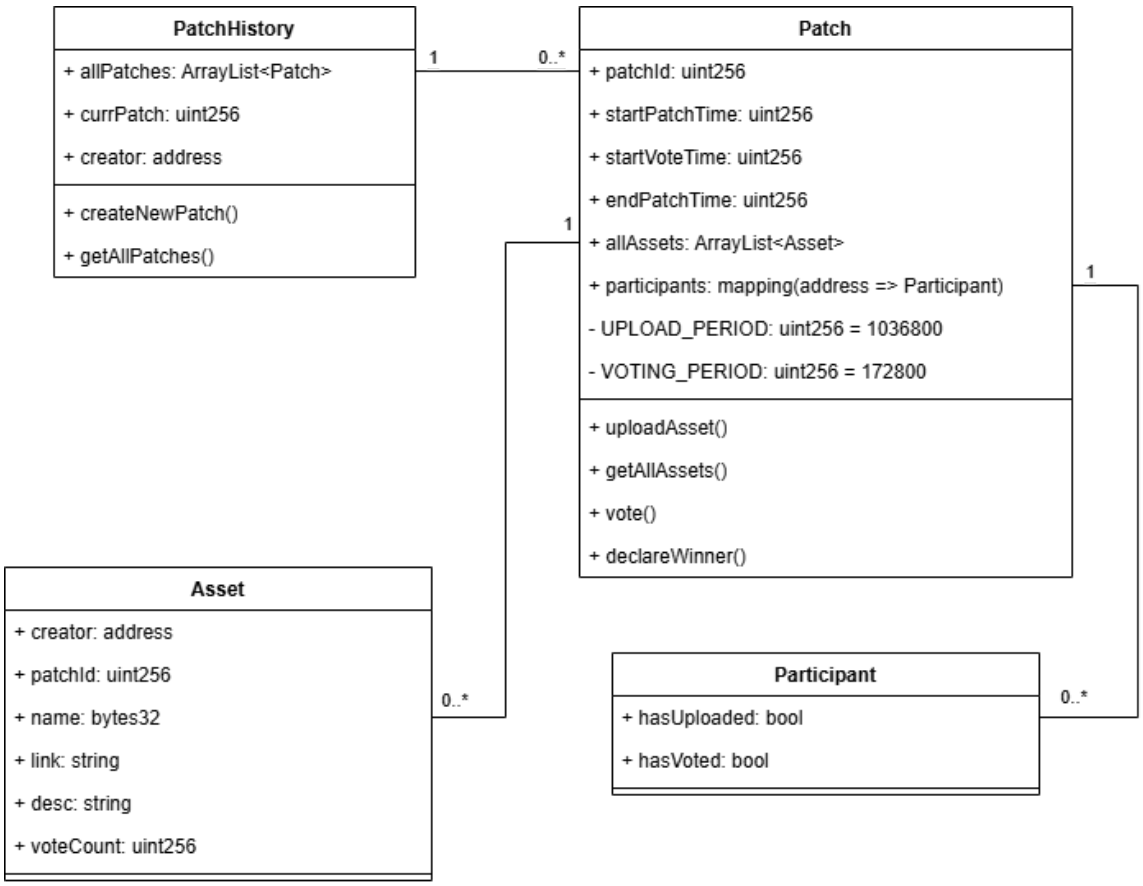
Sistem ini terdiri dari tiga komponen utama: ***Frontend* (FE)**, ***Backend*/Oracle**, dan ***Smart Contract* (SC)**.

1. ***Frontend* (FE)**:
    - Pengguna dapat menghubungkan *wallet* mereka langsung tanpa melibatkan *Backend*.
    - Pengguna melihat data aset dengan meminta informasi dari *Backend* yang diambil dari *Smart Contract*.
    - Untuk voting, pengguna mengirimkan suara melalui *Frontend*, diteruskan ke *Smart Contract* melalui *Backend*/Oracle.
    - Setelah voting selesai, *Frontend* memanggil *Backend* untuk menjalankan fungsi declare winner pada Smart Contract dan menampilkan hasilnya.
2. ***Backend*/Oracle**:
    - Berperan sebagai penghubung antara *off-chain* (misalnya, Google Drive) dengan *Smart Contract*.
    - Untuk pengajuan aset, *Backend* mengunggah file sementara ke Google Drive, kemudian menyimpan link file di blockchain melalui Smart Contract.

- Backend juga meneruskan data suara dari pengguna ke Smart Contract, mengambil hasil voting, dan mengunggah aset pemenang ke repository GitHub.

3. ***Smart Contract* (SC)**:
   - Menyimpan data aset dan hasil voting.
   - Menyediakan fungsi untuk melihat aset, melakukan voting, dan menentukan pemenang (*declare winner*).
   - Berfungsi sebagai pusat logika utama untuk berinteraksi dengan *Frontend* dan *Backend*.

Property dan *Method Smart Contract*



Gambar 3. *Class Diagram Smart Contract*.

PatchHistory : kelas yang dapat membuat dan menampung seluruh *patch* gim yang bersangkutan.

Atribut atau Metode	Deskripsi
`allPatches`	Daftar semua Patch yang pernah dibuat.
`currPatch`	Indeks Patch yang sedang berjalan saat ini.
`creator`	Address admin yang dapat membuat Patch baru.
`createNewPatch()`	Fungsi untuk membuat Patch baru.
`getAllPatches()`	Fungsi untuk mendaftarkan keseluruhan daftar Patch.

Patch : kelas yang merepresentasikan satu *patch* (ronde *upload* dan *voting*) dalam gim.

Atribut atau Metode	Deskripsi
`patchId`	ID Patch yang bersangkutan.
`startPatchTime`	Waktu mulai Patch.
`startVoteTime`	Waktu mulai voting pada Patch yang bersangkutan.
`endPatchTime`	Waktu Patch berakhir.
`allAssets`	Semua Asset yang dikirim pada Patch yang bersangkutan.
`participants`	Mapping *address wallet* dengan status apakah sudah *voting* dan sudah *upload* Asset.
`UPLOAD_PERIOD`	Konstanta lama periode *upload*.
`VOTING_PERIOD`	konstanta lama periode *voting*.
`uploadAsset()`	Fungsi untuk mengirim Asset baru ke dalam *blockchain*.
`getAllAssets()`	Fungsi untuk mendapat keseluruhan daftar Asset dalam Patch.
`vote()`	Fungsi untuk memberikan suara kepada salah satu Asset.
`declareWinner()`	Fungsi untuk mendapatkan pemenang Patch.

Asset : kelas yang merepresentasikan aset gim.

Atribut atau Metode	Deskripsi
`creator`	*Address wallet* pengirim Asset.
`patchId`	ID Patch yang memiliki Asset yang bersangkutan.
`name`	Nama Asset.
`link`	Link ke Google Drive Asset yang bersangkutan.
`desc`	Deskripsi Asset yang bersangkutan.
`voteCount`	Jumlah suara yang dimiliki Asset yang bersangkutan.

Participant : kelas yang merepresentasikan status *upload* dan *voting* peserta.

Atribut atau Metode	Deskripsi
`hasUploaded`	Menandakan apakah peserta sudah *upload* aset atau belum.
`hasVoted`	Menandakan apakah peserta sudah memberikan suara atau belum.

Implementasi *Oracle*

1. **Deskripsi dan Struktur Data**

   Oracle dalam sistem ini berfungsi sebagai penghubung antara sumber data eksternal (Google Drive) dan blockchain melalui *Smart Contract.* Struktur data yang diambil meliputi:

   a. Data aset

      Data aset terdiri atas nama aset, deskripsi aset, *link google drive* aset, dan waktu *upload*.

   b. Data voting

      Data voting terdiri atas indeks aset yang dipilih, *address* pemilih, dan waktu *voting*.

2. **Mekanisme Pengambilan Data**

   Interaksi yang ada pada oracle adalah sebagai berikut:

   a. Pengunggahan aset

      Pengguna mengunggah *file* sementara ke Google Drive melalui Oracle. Kemudian, Oracle mengambil *link file* dari Google Drive dan menyimpannya di blockchain melalui *Smart Contract* menggunakan fungsi **`uploadAsset`**.

   b. Voting aset

      Pengguna memberikan suara untuk aset melalui *Frontend*. Oracle menerima data *voting* dan mengirimnya ke *Smart Contract* menggunakan fungsi **`vote`**.

   c. Penentuan pemenang

      Setelah periode *voting* selesai, *Frontend* memanggil *Backend* untuk mengeksekusi fungsi **`declareWinner`** pada *Smart Contract*. Oracle mengambil hasil *voting* dan *backend* akan menyimpan data pemenang ke *repository* GitHub.

3. **Skenario Buruk dan Solusi**

   a. Skenario Buruk: Gagal Mengestimasi *Gas Fee*

      Penyebabnya adalah akun pengirim tidak memiliki saldo ETH yang cukup untuk membayar *gas fee*. Solusi yang dapat dilakukan adalah melakukan validasi saldo sebelum memulai transaksi dan tampilkan pesan *error* yang jelas kepada operator.

   b. Skenario Buruk: Saldo Akun Tidak Cukup untuk Transaksi

      Penyebabnya adalah kontrak memiliki logika kompleks atau kondisi khusus sehingga estimasi gas tidak dapat dihitung. Solusi yang dapat dilakukan adalah menangkap error saat estimasi gas gagal dan tampilkan *log* untuk *debug*.

c. Skenario Buruk: Gagal Mengirim Transaksi ke *Smart Contract*
   Kemungkinan penyebabnya adalah data transaksi tidak valid atau *node* blockchain sedang tidak responsif. Solusi yang dapat dilakukan adalah implementasi retry mechanism dan validasi parameter transaksi.

Design Pattern

***Smart contract* :**

Factory pattern karena diperlukan satu objek utama yang mampu menyimpan dan membuat beberapa *patch* aplikasi.

***Oracle* :**

Proxy pattern karena Oracle bertindak sebagai penghubung antara sistem eksternal (Google Drive, GitHub) dan *Smart Contract*. *Adapter pattern* karena Oracle juga menyesuaikan format data agar sesuai dengan kebutuhan *Smart Contract* (Misal, konversi nama aset menjadi format hexadecimal).

Anggota Kelompok

Nama (NIM)	Tugas
Salomo Reinhart Gregory Manalu (13521063)	*Backend*, *Oracle*, Integrasi *oracle* dengan *smart contract*, Google Drive, Github
Michael Leon Putra Widhi (13521108)	*Frontend* dan Integrasi *oracle* dengan *frontend*
Nathan Tenka (13521172)	*Smart contract*

Referensi

https://govdevsecopshub.com/2020/11/05/the-challenges-of-open-source-software-and-how-to-solve-them/