PROJECT REPORT
Tugas Besar IF4035 Blockchain
Kickstarter: Implementasi Aplikasi *Crowdfunding* Berbasis Blockchain

Kelompok C

Rachel Gabriela Chen	13521044
Angela Livia Arumsari	13521094
Rinaldy Adin	13521134

PRODI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

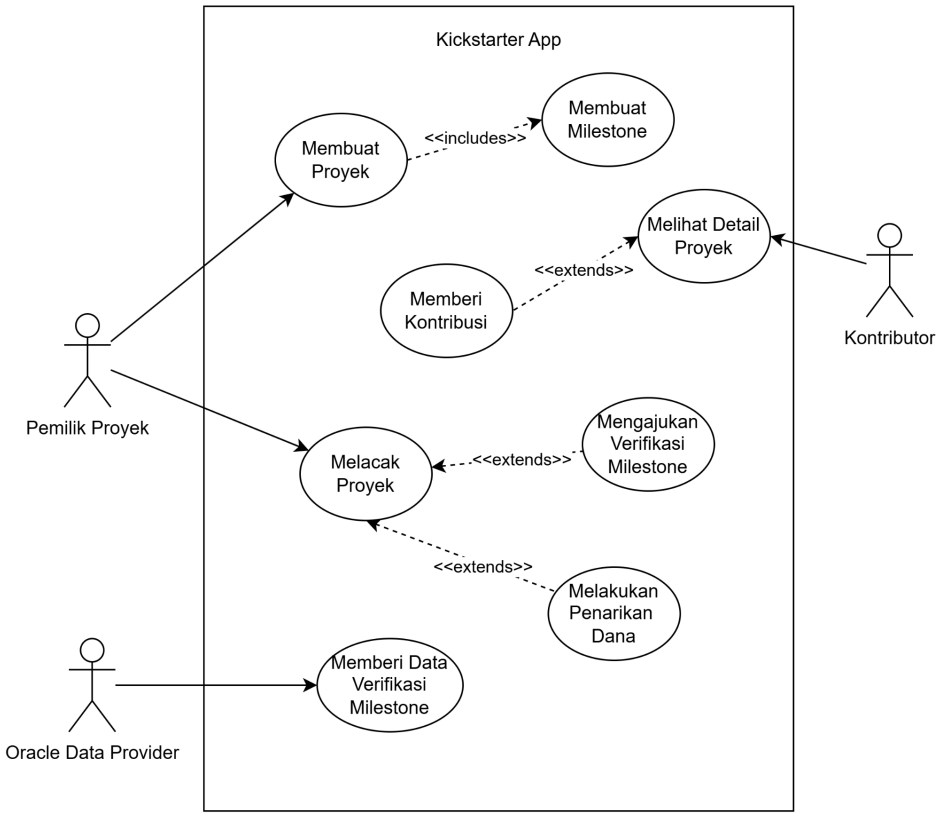
Daftar Isi

Daftar Gambar

I. Problem Statement dan Use Case

Kickstarter adalah aplikasi *crowdfunding* berbasis blockchain yang dirancang untuk mengatasi tantangan utama dalam pengelolaan dana proyek. Masalah utama dalam platform *crowdfunding* tradisional adalah kurangnya transparansi dan akuntabilitas dalam pengelolaan dana, yang sering menimbulkan keraguan dari para kontributor terkait apakah dana yang mereka berikan digunakan sesuai tujuan. Selain itu, risiko penyalahgunaan dana semakin besar karena tidak adanya mekanisme yang memastikan bahwa dana hanya dapat diakses setelah tujuan tertentu tercapai. Dengan memanfaatkan teknologi blockchain, Kickstarter memastikan transparansi, keamanan, dan akuntabilitas dalam setiap transaksi, sehingga semua pihak dapat melacak penggunaan dana dan kemajuan proyek secara *real-time* tanpa perantara.

Berikut adalah *use case diagram* untuk aplikasi Kickstarter.



Gambar 1 *Use Case Diagram*

Pada Kickstarter, terdapat tiga aktor utama. Pemilik proyek dapat membuat proyek beserta *milestone* di dalamnya. Setelahnya, dengan melacak proyek yang dimilikinya, pemilik proyek dapat mengajukan verifikasi *milestone* dan melakukan penarikan dana. Kontributor dapat melihat detail proyek yang tersedia pada Kickstarter dan memberikan kontribusi. *Oracle data provider* adalah pihak ketiga yang akan menyediakan data verifikasi milestone ke aplikasi Kickstarter.

II. Platform Blockchain

Platform blockchain yang dipilih untuk aplikasi Kickstarter adalah Geth, yaitu implementasi dari Ethereum berbasis Golang yang bersifat *open-source*. Ethereum mendukung *smart contracts* yang memungkinkan otomatisasi dan transparansi dalam pelaksanaan *milestone* proyek.

Private Ethereum dipilih atas dasar kebutuhan untuk menjaga data proyek dan transaksi tetap terkontrol dan hanya dapat diakses oleh pihak yang relevan, seperti pengguna yang terlibat dalam proyek. Selain itu, penggunaan Solidity sebagai bahasa pengembangan *smart contracts* menawarkan familiaritas bagi pengembang, didukung komunitas luas, dan dokumentasi lengkap. Dengan kombinasi otomatisasi, privasi, dan dukungan teknis, *private Ethereum* menjadi pilihan utama platform blockchain untuk aplikasi ini.

III. Tech Stack

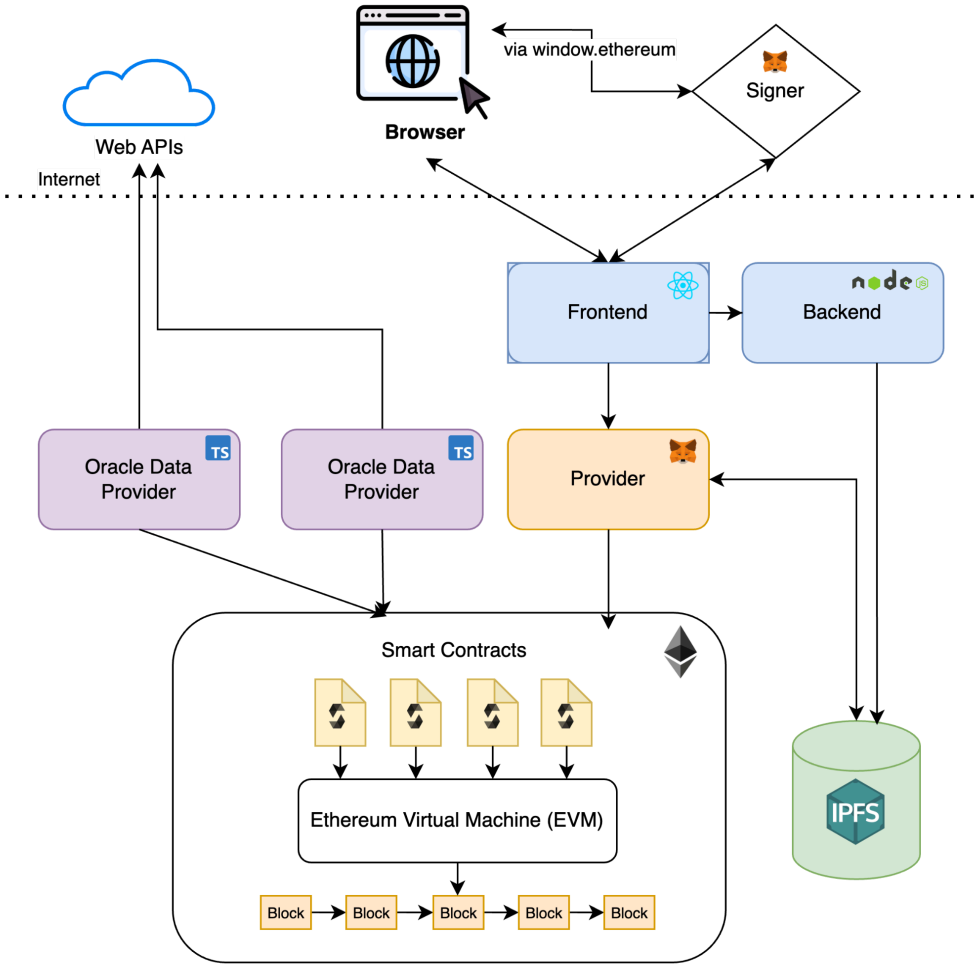
Berikut adalah teknologi yang digunakan dalam pembangunan Kickstarter.

1. **Platform Blockchain - Geth**
   Alasan pemilihan platform blockchain telah dijelaskan pada bagian sebelumnya.
2. **Network Deployment - Kurtosis**
   Kurtosis digunakan untuk mempermudah proses pengelolaan jaringan blockchain privat.
3. **Smart Contract - Solidity**
   Solidity dipilih karena kompatibilitas dengan Ethereum. Selain itu, solidity juga didukung komunitas besar sehingga mempermudah proses pengembangan aplikasi.
4. **Kerangka Pengujian (Local) - Hardhat**
   Hardhat digunakan untuk mempermudah pengembangan lokal karena memiliki fitur lengkap, seperti *debugging*, simulasi jaringan, dan otomatisasi pengujian.
5. **Penyedia Data Oracle - TypeScript**
   TypeScript digunakan karena mendukung tipe data, pengembangan yang lebih terstruktur, dan integrasi mudah dengan sistem berbasis *event-driven* untuk pengelolaan data oracle.
6. **Backend IPFS - ExpressJS**
   ExpressJS digunakan karena merupakan *framework* ringan yang mendukung penggunaan npm untuk integrasi dengan *web3.storage*. Implementasinya cepat dan cocok untuk kebutuhan *backend* sederhana.
7. **IPFS - web3.storage**
   Web3.storage dipilih karena menyediakan penyimpanan terdesentralisasi dengan opsi gratis. Hal ini mendukung kebutuhan penyimpanan data untuk aplikasi berbasis blockchain.
8. **Frontend - React.js**
   React.js dipilih karena familiaritas oleh pengembang. Selain itu, React juga mendukung pembuatan antarmuka pengguna yang interaktif serta kompatibel dengan ekosistem dApps.

9. **Penyedia Wallet - Metamask**
   Metamask digunakan untuk mempermudah pengguna dalam berinteraksi dengan jaringan blockchain.

IV. *High Level* Desain

Berikut adalah high level desain dari kickstarter yang terdiri dari berbagai komponen.



Gambar 2 *High Level* Desain

Setiap komponen pada Kickstarter memiliki perannya masing-masing. Berikut adalah penjelasan dari tiap komponen.

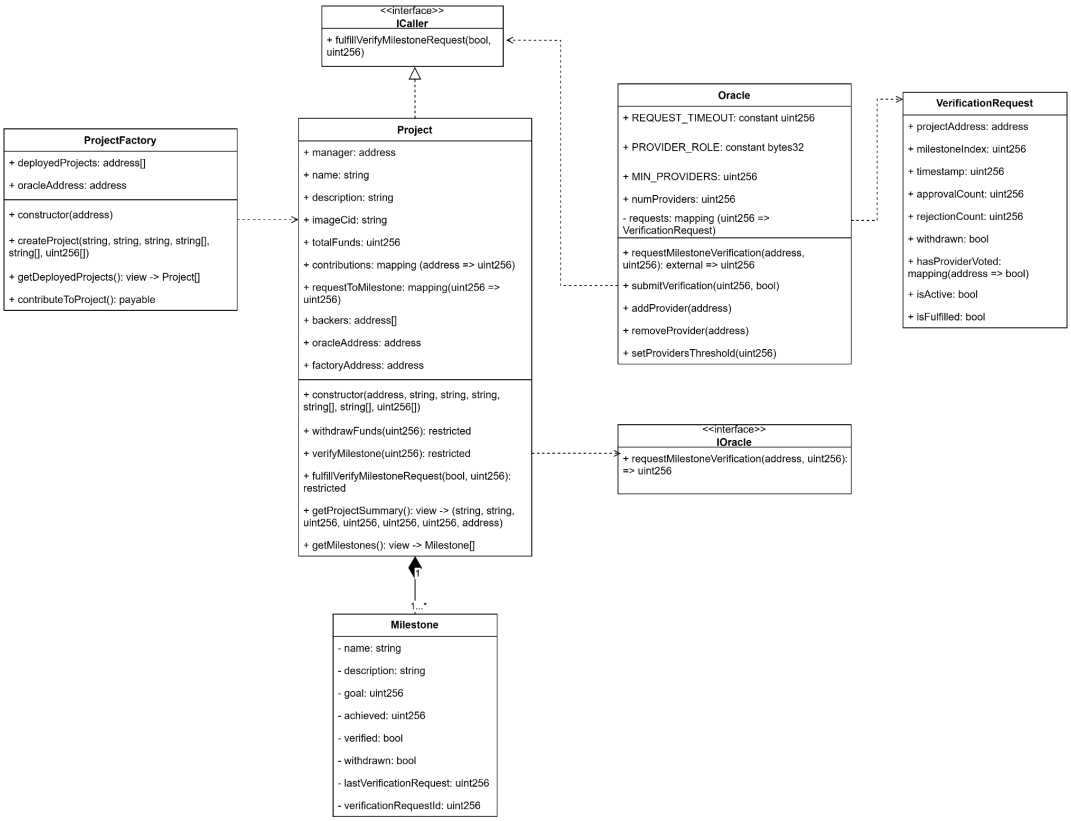
1. **Browser & Signer**
   Browser bertindak sebagai antarmuka pengguna utama, dengan pengguna menggunakan *wallet provider* Metamask (melalui `window.ethereum`) untuk melakukan *sign* dan interaksi dengan smart contract.
2. **Frontend**
   Komponen *frontend* menggunakan React.js untuk menyediakan antarmuka

pengguna yang dinamis dan interaktif, memfasilitasi akses ke fitur aplikasi seperti melihat status proyek dan melakukan kontribusi.

3. ***Backend***
   Backend menggunakan Node.js (ExpressJS) untuk menangani logika pengelolaan data yang akan disimpan di IPFS. *Backend* dibutuhkan karena *upload data* ke IPFS membutuhkan *client* dan *agent* dengan *capability* melakukan *upload* data ke *space* di IPFS. *Client* dan *agent* ini dibuat dengan *key* dan *proof* yang disimpan sebagai *environment variables*. Karena penggunaan data yang sensitif, *frontend* akan melakukan *request* lewat *backend* untuk *upload* data ke IPFS.
4. **Provider (Metamask)**
   Metamask bertindak sebagai penghubung antara pengguna dan blockchain, memfasilitasi transaksi dan akses ke Ethereum Virtual Machine (EVM).
5. **Smart Contracts**
   Smart contracts yang ditulis dalam Solidity menjalankan logika aplikasi, seperti mengelola milestone proyek, pencairan dana, dan validasi transaksi di blockchain Ethereum.
6. **Ethereum Virtual Machine (EVM)**
   EVM adalah lapisan eksekusi untuk *smart contracts*, bertanggung jawab untuk memastikan logika aplikasi berjalan sesuai dengan aturan blockchain.
7. **Oracle Data Provider**
   Komponen ini berfungsi untuk menyediakan data dari sumber eksternal ke *smart contracts*. Oracle memungkinkan validasi data proyek secara otomatis dari API pihak ketiga.
8. **IPFS**
   IPFS digunakan untuk menyimpan data proyek secara terdesentralisasi. *Web3.storage* mempermudah pengelolaan file ke IPFS, memberikan solusi penyimpanan yang aman dan gratis.

V. Smart Contract

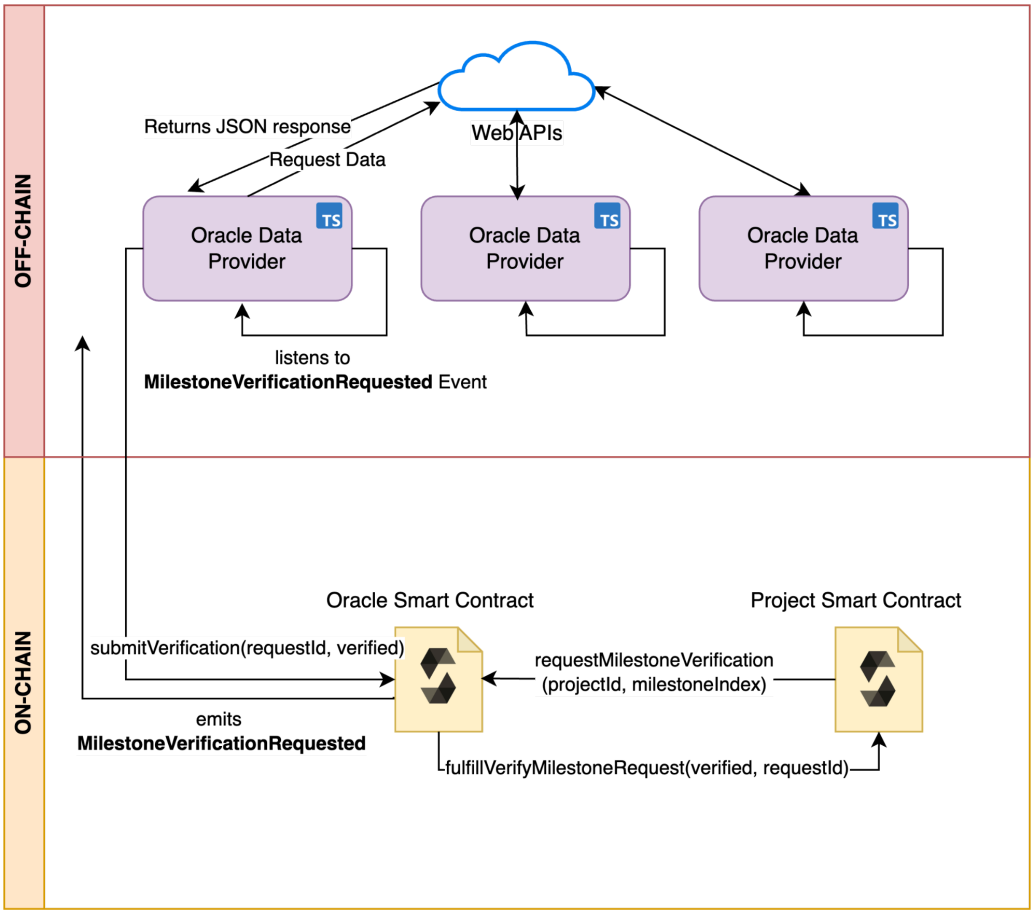
Berikut adalah properti dan *method* yang ada pada smart contract.



Gambar 3 *Class Diagram*

VI. Implementasi Oracle

Berikut adalah flow verifikasi dengan menggunakan Oracle.



Gambar 4 Flow Verifikasi Oracle

Secara umum, Oracle terbagi menjadi dua, yaitu Off-Chain Oracle yang bertugas untuk mengumpulkan data dari Web API dan On-Chain Oracle. Data yang dikumpulkan dari Web API berupa *random number* untuk memudahkan implementasi. *Random number* yang berupa angka genap berarti bahwa *milestone* telah selesai, dan sebaliknya jika ganjil.

Saat user melakukan "Request Verification" melalui Web UI, maka akan dipanggil fungsi `verifyMilestone` pada Project Smart Contract. Fungsi ini kemudian meminta verifikasi dari Oracle Smart Contract yang akan men-*generate* sebuah `VerificationRequest` dengan ID berdasarkan proyek dan milestone yang akan diverifikasi. Kemudian, Oracle SC akan meng-*emit* event MilestoneVerificationRequested yang akan di-*listen* oleh Off-Chain Oracle Data Provider. Off-Chain Oracle Data Provider kemudian akan melakukan *request* ke Web API (dalam hal ini *random number generator*) untuk mendapatkan data verifikasi. Setelah itu, data yang didapatkan akan di-*submit* ke Oracle SC.

Jika jumlah *provider* yang melakukan *submit* data sudah mencapai *threshold*, maka akan dilakukan pengecekan hasil akhir berdasarkan hasil data mayoritas. Hasil inilah yang akan dikembalikan ke Project SC melalui fungsi `fulfillVerifyMilestoneRequest`.

Beberapa skenario buruk yang bisa terjadi yaitu:

1. Salah satu Off-Chain Oracle Node *down*. Hal ini ditangani dengan menyimpan MIN_PROVIDERS pada Oracle SC. Artinya, terdapat beberapa *provider* yang dapat melakukan *submit* data. Selama jumlah *provider* masih memenuhi MIN_PROVIDERS, maka sistem tetap dapat berjalan dengan lancar.
2. Double voting oleh Off-Chain Oracle Data Provider. Hal ini dimitigasi dengan menyimpan data Oracle yang telah melakukan *voting* pada struct VerificationRequest, sehingga dipastikan tidak ada *provider* yang melakukan *double voting*.
3. Oracle Manipulation Attack. Hal ini ditangani dengan data provider yang terdesentralisasi seperti yang dijelaskan pada no.1.

VII. Design Pattern

Berikut beberapa *design pattern* yang digunakan dalam implementasi smart contract dan oracle.

1. Factory Pattern
   Smart contract ProjectFactoy merupakan *factory* terpusat untuk membuat kontrak Project. Alasan pemilihan *pattern* ini yaitu untuk kemudahan pembuatan dan pelacakan proyek. Seluruh pembuatan kontrak Project akan melalui ProjectFactory dan dapat dilacak dengan metode `getDeployedProjects()`. Hal ini juga memastikan Project selalu dibuat dengan parameter yang konsisten.
2. State Pattern
   State pattern digunakan pada kontrak Project untuk mengelola milestone dengan transisi status yang jelas (unverified → verified → withdrawn). Pola ini memastikan integritas data melalui transisi status yang atomik dan mengikuti aturan tertentu, serta mendukung auditabilitas dengan kemudahan pelacakan status dan riwayat setiap milestone.
3. Role-Based Access Control Pattern
   Dalam implementasi Oracle, digunakan Role-Based Access Control pattern yang diimplementasikan melalui AccessControl dari OpenZeppelin. Hal ini digunakan untuk memastikan hanya *address* tertentu yang dapat melakukan operasi sensitif.
4. Observer Pattern
   Implementasi Oracle menggunakan *event-driven observer pattern* untuk melakukan observasi terhadap permintaan verifikasi. Penggunaan observer ini memisahkan permintaan verifikasi dari pemrosesannya dan memungkinkan provider untuk merespons permintaan secara paralel.

VIII. Optimasi Smart Contract

Berikut beberapa optimasi yang digunakan dalam smart contract yang diimplementasi

1. Memanfaatkan *off-chain storage* berupa emit Events
   Dengan memanfaatkan emit events untuk menyimpan data mengenai *events* yang terjadi, seperti mencatat adanya kontribusi yang dilakukan. Dengan

memanfaatkan emit events, yang termasuk dalam off-chain storage dalam ethereum, gas cost untuk penyimpanan data yang read-only dapat dioptimasi, seperti pada penyimpanan timestamp pada suatu kontribusi. Hal tersebut karena data yang read-only dan tidak perlu digunakan dalam logika smart contract lebih baik disimpan dengan events karena penyimpanan ke dalam *storage* atau *field* smart contract akan menggunakan gas cost yang lebih besar.

2. Memanfaatkan modularitas dengan factory design pattern
   Dengan adanya contract ProjectFactory yang menghasilkan contract Project yang baru untuk setiap proyek baru yang ingin dibuat pengguna, implementasi yang dibuat menjadi lebih optimal dengan adanya *separation of concerns* antara masing-masin Project yang pernah dibuat.
3. Memanfaatkan keyword storage untuk update data milestone
   Dengan memanfaatkan keyword storage pada fungsi yang melakukan perubahan terhadap array milestone seperti receiveContribution dan withdrawFunds, fungsi tersebut akan melakukan akses dan perubahan secara langsung terhadap data array milestone. Tanpa memanfaatkan keyword storage, data milestone perlu di-copy terlebih dahulu ke *local variable* pada fungsi, menyebabkan gas cost yang lebih besar.

IX. Pembagian Tugas

Nama	**NIM**	**Tugas**
Rachel Gabriela Chen	13521044	- Implementasi Oracle SC - Implementasi Off Chain Oracle - Implementasi Withdraw Funds
Angela Livia Arumsari	13521094	- Integrasi dengan wallet - Implementasi display projects - Implementasi integrasi dengan IPFS
Rinaldy Adin	13521134	- Inisialisasi development environment dengan hardhat, serta inisialisasi dan setup deployment menggunakan geth dan kurtosis. - Implementasi awal smart contract project dan project factory. - Implementasi kontribusi project