Tugas Besar IF4033 Blockchain
Implementasi Aplikasi Berbasis Blockchain
Semester 1 Tahun 2024/2025

Disusun oleh:
Kelompok K

Yanuar Sano Nur Rasyid	13521110
Rayhan Hanif Maulana Pradana	13521112
Muhammad Naufal Nalendra	13521152

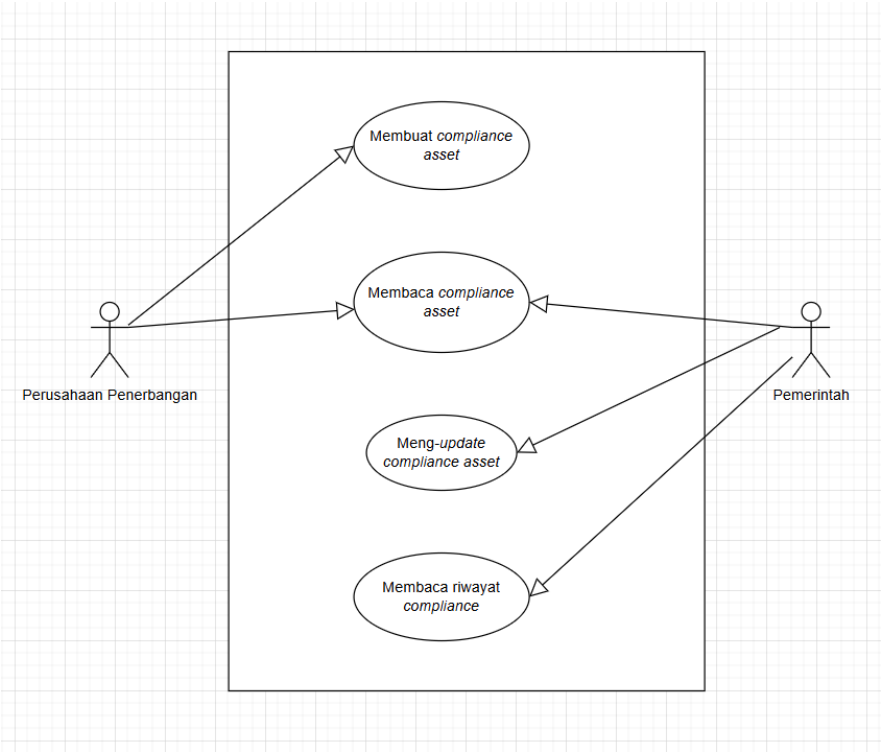
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

Problem Statement

Transportasi merupakan hal yang sudah tidak asing lagi dalam kehidupan sehari-hari. Perkembangan transportasi memungkinkan kita untuk berpindah dari satu tempat ke tempat lain dengan cepat, mudah, dan aman. Salah satu hal penting dalam pengaman transportasi adalah sertifikasi keamanan untuk transportasi aviasi atau penerbangan. Proses verifikasi tersebut dilakukan dengan mengirimkan data performa serta *compliance record* dari perusahaan terhadap pihak yang berwenang seperti pemerintah. Akan tetapi, proses verifikasi tersebut memiliki beberapa tantangan, seperti adanya ketergantungan data yang diberikan oleh pihak perusahaan, adanya risiko manipulasi data dari laporan yang dikirimkan, serta kurangnya transparansi dalam proses penyimpanan dan pengelolaan informasi. Hal tersebut dapat menghambat proses verifikasi penerbangan sehingga dapat menimbulkan kerugian. Oleh karena itu, diperlukan solusi inovatif yang dapat menghadapi tantangan-tantangan tersebut.

Salah satu solusi terhadap masalah tersebut adalah dengan menggunakan teknologi *blockchain. Blockchain* adalah teknologi yang memungkinkan pencatatan data terdistribusi dalam bentuk *distributed ledger* yang saling menyambung dan diamankan dengan kriptografi. Fitur dari *blockchain* antara lain adalah desentralisasi, transparansi, keamanan, dan *immutable.* Dengan menggunakan *blockchain*, aplikasi yang melakukan verifikasi terhadap laporan dan catatan dari perusahaan penerbangan dapat dibuat sebagai solusi dari tantangan sebelumnya.

Aplikasi AviationComplianceDApp dibuat sebagai aplikasi verifikasi dan konfirmasi terhadap *compliance* suatu perusahaan penerbangan dengan aturan serta sebagai solusi dari tantangan kebergantungan terhadap perusahaan, risiko manipulasi, dan transparansi proses. *Use case* dari aplikasi ini antara lain membuat *compliance asset,* membaca *compliance asset,* meng-*update compliance asset,* serta mengakses riwayat *compliance asset*.



Gambar 1 Use Case Diagram

Platform Blockchain

Platform *blockchain* yang dipilih untuk pembuatan *distributed* aplikasi adalah Hyperledger Fabric. Hyperledger Fabric adalah sebuah platform *blockchain permisioned* yang berarti hanya pihak yang diperbolehkan yang bergabung serta memiliki *use case* khusus untuk *enterprise*. Platform fabric dipilih karena fitur-fitur yang disediakan cocok dengan *problem statement* yang sebelumnya sudah diberikan. Fitur yang dimaksud adalah modularitas yang memungkinkan fitur-fitur yang digunakan dalam jaringan sesuai dengan kebutuhan perusahaan, contohnya adalah *consensus protocol* yang digunakan, *membership* dan *ordering* service, serta *peer to peer gossip* yang digunakan. Selain itu, *smart contract* yang disebut *chaincode* mendukung *general purpose programming language*, seperti Java, Go, dan Node.js dan tidak bergantung dengan *domain specific language* (DSL). Selain itu, Fabric tidak memerlukan *cryptocurrency* untuk menjalankan *smart contract* serta jaringan yang *permissioned* cocok dengan *use case* aplikasi.

Tech Stack

Tech Stack yang digunakan adalah Hyperledger Fabric sebagai *platform blockchain,* Fabric Gateway sebagai *backend,* Vue sebagai *frontend,* serta Golang sebagai Oracle. Pemilihan Fabric Gateway karena memiliki dukungan langsung dengan platform Fabric serta memiliki dukungan

bahasa pemrograman yang baik mencakup Go, Node.js, dan Java sehingga memudahkan pengembang saat membuat *backend* untuk Fabric.

Pemilihan Vue karena memiliki performa yang baik sebagai *frontend* karena *lightweight* dan menggunakan *virtual DOM* yang memungkinkan UI memiliki respons yang cepat. Selain itu, Vue memiliki dokumentasi dan *support* dari komunitas, seperti *library* yang baik sehingga memudahkan pengembangan fitur-fitur *frontend.*

Pemilihan Golang didasari oleh performa yang tinggi dan efisien yang cocok untuk digunakan dalam *blockchain.* Selain itu, Golang memiliki ekosistem luas yang dapat mendukung dalam pembuatan *oracle* karena terdapat *library* pendukung. Terakhir, bahasa pemrograman yang sama dengan *backend* yang menggunakan Fabric Gateway memudahkan penggabungan antara *oracle* dengan jaringan *blockchain*.

High Level Design

Komponen yang terlibat dalam dApps adalah jaringan *blockchain, backend, oracle,* dan *frontend.* Dalam jaringan *blockchain*, terdapat *node* dan komponen pendukung, seperti *peer node* sebagai *node* utama jaringan Fabric dan penyimpan *ledger* serta pengeksekusi transaksi melalui *chaincode* dalam jaringan. Kemudian, *orderer node* sebagai pengurut transaksi yang terjadi dalam jaringan *blockchain*. *Node* ini diperlukan karena Fabric menggunakan *deterministic consensus algorithm* yang berbeda dari *probabilistic consensus algorithm* yang digunakan oleh Ethereum dan Bitcoin. Selain itu. terdapat *chaincode* yang berfungsi sebagai yang berisi *smart contract* mengenai bagaimana logika transaksi dilakukan serta bagaimana cara men-*deploy smart contracts* tersebut. Terakhir, terdapat *channel* antara node yang berfungsi sebagai sebuah *subnet* komunikasi antara anggota jaringan yang bersifat *private* dan *confidential* untuk melakukan sebuah transaksi.

Dalam *backend,* terdapat komponen *wallet* yang berfungsi sebagai penyimpan identitas pengguna dalam jaringan Fabric. *Wallet* memiliki tiga jenis yang bergantung dari cara penyimpanan identitasnya, yaitu *file system, memory,* dan *database*. Pada aplikasi jenis *wallet* yang digunakan adalah *file system* yang dihasilkan oleh *cryptogen* dengan jenis identitas yang digunakan adalah X509Identity. Setelah menentukan identitas, *backend* juga memiliki komponen yang tersambung dengan jaringan *blockchain* Fabric yaitu menggunakan *library* Fabric Gateway yang memiliki dukungan langsung untuk terhubung dengan jaringan Fabric. Bahasa yang digunakan dalam Fabric Gateway adalah Go. Gateway ini berfungsi juga sebagai jalur komunikasi antara *blockchain* dengan *frontend*.

Dalam *oracle,* terdapat komponen FlightData yang berfungsi untuk menyimpan data penerbangan yang diperoleh dari API eksternal. Data tersebut diambil oleh fungsi FetchFlightData. FetchFlightData akan melakukan *request* ke *endpoint flights* aviationstack API

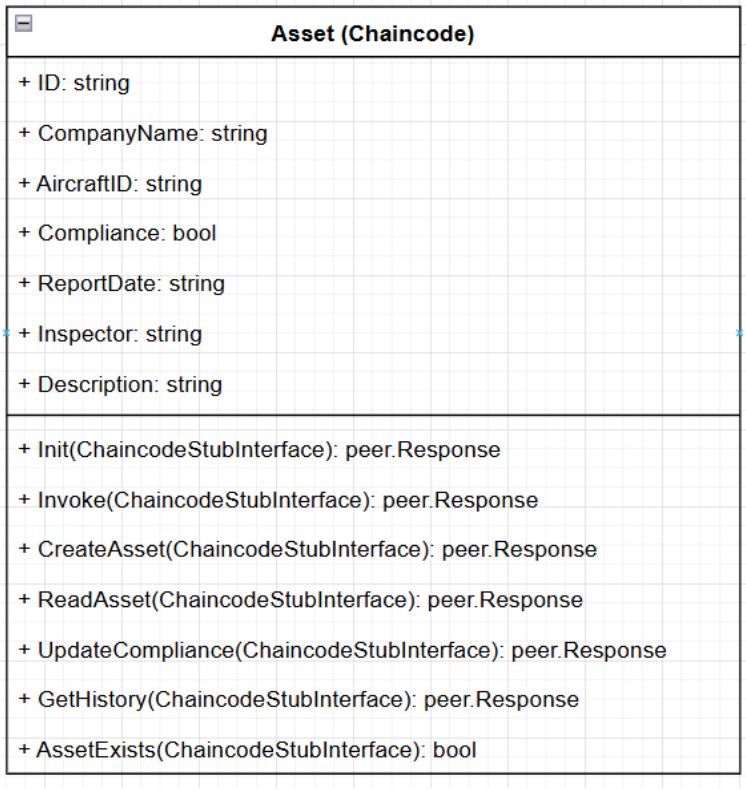
menggunakan sebuah API-key dan *query param* flight_iata untuk mendapatkan data penerbangan dengan ID penerbangan yang dipilih.

Pada *frontend,* digunakan framework Vue untuk NodeJs dengan bahasa typescript untuk menciptakan UI yang modular dan interaktif. Framework Vue khususnya memungkinkan pengembangan aplikasi web dengan basis komponen. Vue juga menyediakan fleksibilitas dengan dua jenis API yang bisa digunakan serta template HTML yang bisa dipasang dengan berbagai script dan styling. Selain template Vue dasar sebagai root component, terdapat lima buah komponen yang menjadi children dari root component. Kelima komponen tersebut adalah Create Asset, Read Asset, Asset History, Update Compliance, dan Wallet Setting. Setiap komponen menyediakan fitur yang sesuai dengan backend API terkait yang nantinya akan di-consume menggunakan Axios

Smart Contract

Smart contract atau *chaincode* yang dibuat memiliki beberapa properti dan *method*. Penjelasan dari properti Asset (Chaincode) yaitu ID sebagai identifikasi unik setiap aset, CompanyName sebagai nama perusahaan terkait aset, AircraftID sebagai identifikasi pesawat terkait aset, Compliance sebagai status kepatuhan aset, ReportDate sebagai tanggal laporan aset, Inspector nama inspektor yang melakukan pengecekan, Deskription sebagai deskripsi lanjut dari sebuah aset.

Metode yang terdapat dalam *chaincode,* antara lain Init sebagai fungsi menginisialisasi data *ledger* yang dijalankan saat *chaincode* diinstall, Invoke fungsi utama yang menangani saat terjadi transaksi di *chaincode*. Fungsi yang dapat dipanggil oleh Invoke, antara lain CreateAsset yaitu fungsi untuk membuat aset baru dalam *ledger,* ReadAsset yaitu fungsi untuk membaca data aset dalam *ledger,* UpdateCompliance yaitu fungsi untuk mengubah data *compliance* dalam suatu aset, GetHistory yaitu fungsi untuk mendapatkan riwayat perubahan suatu aset. Terhakhir, terdapat fungsi AssetExists sebagai fungsi pembantu yang melakukan pengecekan apakah sebuah aset sudaha ada di *ledger* atau belum.



Gambar 2 Class Diagram *Chaincode*

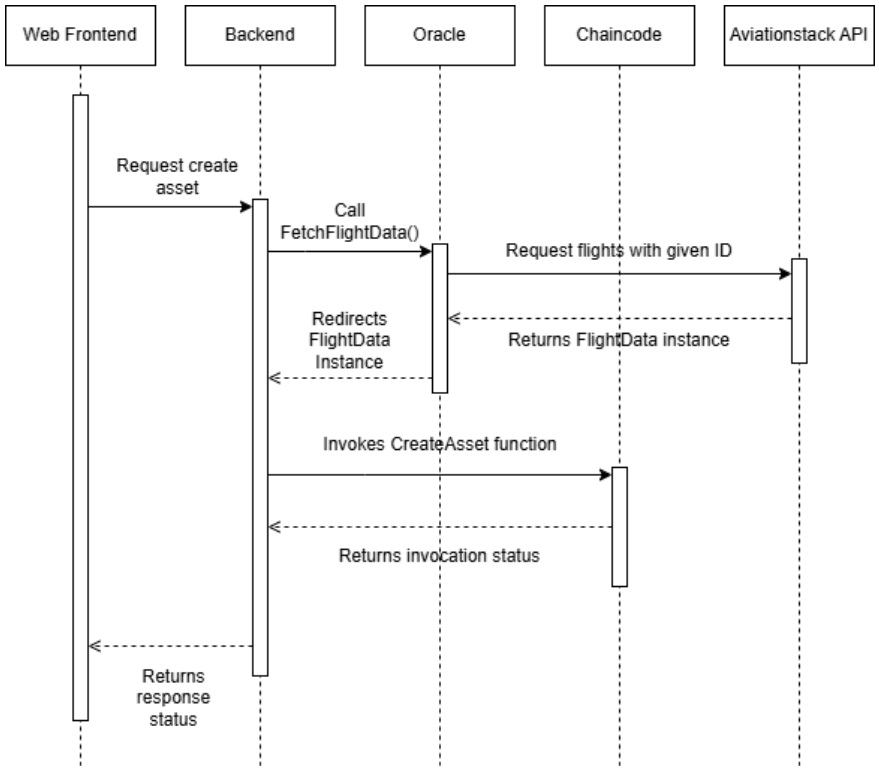
Oracle

Data *offchain* yang diambil oleh oracle digunakan untuk melengkapi data *asset* yang dibuat oleh pemanggilan fungsi *chaincode*. Dalam *chaincode*, didefinisikan *asset* dengan atribut-atribut yang berkaitan dengan informasi sebuah inspeksi pesawat terbang, salah satunya adalah perusahaan yang memiliki pesawat tersebut, yaitu atribut CompanyName. Kepemilikan pesawat dapat divalidasi dengan menggunakan data penerbangan yang didapatkan melalui API aviationstack. Berikut adalah struktur dari data penerbangan.

```
type FlightData struct {
    FlightStatus   string `json:"flight_status"`
    Departure      string `json:"departure"`
    Arrival        string `json:"arrival"`
    FlightNumber   string `json:"flight_number"`
    AirlineName    string `json:"airline_name"`
    AircraftType   string `json:"aircraft_type"`
    DepartureTime  string `json:"departure_time"`
    ArrivalTime    string `json:"arrival_time"`
    DepartureCity  string `json:"departure_city"`
    ArrivalCity    string `json:"arrival_city"`
}
```

Dapat dilihat bahwa terdapat atribut AirlineName, atau nama maskapai, yaitu nama perusahaan yang mengelola pesawat yang terkait. Oleh karena itu, *client* tidak perlu mengirimkan data perusahaan secara manual. Hal tersebut memastikan tidak ada kesalahan dalam kepemilikan pesawat. Data

API eksternal yang digunakan adalah aviationstack API, yang menyediakan data penerbangan secara *real-time*. *Endpoint-endpoint* aviationstack API dapat diakses menggunakan sebuah API-Key yang disediakan. Pengambilan data dilakukan ketika sebuah *asset* akan dibuat dengan memanggil fungsi CreateAsset pada *chaincode*, yang dilakukan oleh *client* melalui *backend*. Sebelum pemanggilan fungsi CreateAsset dilakukan, *backend* akan terlebih dahulu memanggil skrip *oracle* yang akan melakukan *request* eksternal ke salah satu *endpoint* aviationstack API. *Request* tersebut disertai API-Key dan *query param* flight_iata, yang digunakan untuk menemukan data penerbangan dengan ID tertentu. *Response* dalam bentuk JSON kemudian akan diterima oleh *oracle* dan diteruskan ke *backend*. *Backend* akan mem-*parsing response* tersebut dan melempar atribut AirlineName sebagai sebuah argumen untuk fungsi CreateAsset. Fungsi CreateAsset akan membuat sebuah aset dengan atribut CompanyName sesuai AirlineName yang diperoleh. Berikut adalah ilustrasi mekanisme pengambilan data dari aviationstack API dengan *sequence diagram* sederhana.



Gambar 3 Sequence Diagram Pemanggilan Aviationstack API melalui *Oracle*

Terdapat dua skenario buruh yang dapat terjadi, yaitu ketika *response* tidak ditemukan, atau ketika data penerbangan itu sendiri yang tidak ditemukan. *Response* yang tidak ditemukan juga berarti bahwa data penerbangan tidak ditemukan. Dua alternatif solusi yang kami ajukan adalah membuat CompanyName menjadi "Unavailable", atau *abort* transaksi sekaligus. Masalah dari alternatif solusi pertama, yaitu membuat CompanyName menjadi "Unavailable", adalah ketika pesawat sebenarnya dimiliki oleh sebuah maskapai, namun karena masalah, nama maskapai dari pesawat tersebut adalah "Unavailable". Hal ini kembali lagi ke masalah yang awalnya kami hindari, yaitu kesalahan dalam kepemilikan. Alternatif solusi kedua juga menjadi masalah ketika terdapat sebuah pesawat yang tidak dimiliki oleh sebuah maskapai. Akan tetapi, sistem *blockchain* yang kami buat dikhususkan untuk penggunaan *enterprise*, sehingga dapat dipastikan bahwa aset pesawat yang dibuat akan memiliki nama perusahaan, sehingga solusi kedua kami pilih. Berikut adalah implementasi kegagalan pada *oracle*.

```
flightData, err := FetchFlightData(request.AircraftID)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": fmt.Sprintf("Failed to fetch flight data: %v", err)})
        return
    }

    companyName := flightData.AirlineName
```

Implementasi tersebut merupakan implementasi *error-handling* standar dalam bahasa pemrograman Go. FetchFlightData merupakan fungsi *oracle* untuk mengambil data dari aviationstack API, dan mengembalikan data penerbangan, yaitu FlightData, dan error. Ketika error tidak nil, yang artinya terdapat sebuah masalah, baik data tidak ditemukan, atau *endpoint* aviationstack tidak mengirimkan sebuah *response*, maka transaksi akan digagalkan atau *abort*.

Design Pattern

Design pattern yang digunakan oleh implementasi *oracle* kami adalah Request-Response dan Pull-based Inbound Oracle. Pemanggilan yang dimulai dari *client* hingga didapatkan data dari API eksternal melalui *oracle* bersifat konkuren, dengan masing-masing komponen menunggu untuk *response* dari komponen yang dipanggilnya. Pull-based Inbound Oracle merupakan ketika *smart contract* membutuhkan data *offchain* dari *oracle*. Hal tersebut sama seperti kasus *oracle* yang kami buat, yaitu *chaincode* mendapatkan data penerbangan yang dikumpulkan oleh *oracle* ketika CreateAsset akan dieksekusi.

Pembagian Tugas

NIM	Nama	Tugas
13521110	Yanuar Sano Nur Rasyid	Setup Hyperledger Fabric, Wallet
13521112	Rayhan Hanif Maulana Pradana	Backend, Oracle
13521152	Muhammad Naufal Nalendra	Smart Contract, Frontend

Referensi

https://hyperledger-fabric.readthedocs.io/en/release-2.5/

https://hyperledger.github.io/fabric-gateway/

https://vuejs.org/guide/introduction.html

https://medium.com/coinmonks/blockchain-oracle-design-patterns-explained-1ce2e185fa1d

Lampiran

https://github.com/yansans/AviationComplianceDApp

https://drive.google.com/file/d/1SstqAB5AFBSbh2hc_0__7JzNcfeyA5CW/view?usp=sharing