Implementasi RESTful Principles pada Web Pencegahan Kecurangan

Tugas Akhir

diajukan untuk memenuhi salah satu syarat

memperoleh gelar sarjana

dari Program Studi S1 Rekayasa Perangkat Lunak

Fakultas Informatika

Universitas Telkom

1302200028

Rama Padliwinata

Program Studi Sarjana S1 Rekayasa Perangkat Lunak

Fakultas Informatika

Universitas Telkom

Bandung

2024

LEMBAR PENGESAHAN

Implementasi RESTful Principles pada Web Pencegahan Kecurangan

RESTful Implementation in Fraud Deterrence Web

NIM :1302200028
Rama Padliwinata

Tugas akhir ini telah diterima dan disahkan untuk memenuhi sebagian syarat memperoleh gelar pada Program Studi Sarjana S1 Rekayasa Perangkat Lunak
Fakultas Informatika
Universitas Telkom

Bandung, 6 September 2024
Menyetujui

Pembimbing I,	Pembimbing II,
Dr. Arfive Gandhi, S.T., M.T.I.	Muhammad Johan Alibasa, S.T., M.T., Ph. D.
NIP:22910008	NIP:21900001

Ketua Program Studi
Sarjana S1 Rekayasa Perangkat Lunak,

Dr. Mira Kania Sabariah, S.T., M.T.
NIP: 14770011

LEMBAR PERNYATAAN

Dengan ini saya, Rama Padliwinata, menyatakan sesungguhnya bahwa Tugas Akhir saya dengan judul Implementasi RESTful Principles Pada Web Pencegahan Kecurangan beserta dengan seluruh isinya adalah merupakan hasil karya sendiri, dan saya tidak melakukan penjiplakan yang tidak sesuai dengan etika keilmuan yang belaku dalam masyarakat keilmuan. Saya siap menanggung resiko/sanksi yang diberikan jika di kemudian hari ditemukan pelanggaran terhadap etika keilmuan dalam buku TA atau jika ada klaim dari pihak lain terhadap keaslian karya,

Bandung, 6 September 2024

Yang Menyatakan

Rama Padliwinata

Implementasi RESTful Principles pada Modul Pencegahan Kecurangan

Rama Padliwinata[1], Arfive Gandhi[2], Muhamad Johan Alibasa3[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]rpadliwinata@students.telkomuniversity.ac.id, [2]arfivegandhi@telkomuniversity.ac.id,
[3]alibasa@telkomuniversity.ac.id

Abstrak

Fraud Deterrence Propeller (FDP) adalah solusi yang baik untuk mencegah terjadinya kecurangan pada sebuah entitas. Implementasinya yang berupa asesmen juga membuatnya mudah untuk dilakukan oleh entitas kecil sampai besar. Solusi yang baik tersebut masih bisa ditingkatkan dengan cara melakukan digitalisasi FDP. Salah satu cara digitalisasi FDP adalah dengan membuat Web Pencegahan Kecurangan yang mengimpelmentasikan FDP tersebut. Untuk membangun Web Pencegahan Kecurangan yang baik diperlukan kaidah implementasi yang tepat. Salah satu kaidah implementasi yang tepat adalah kaidah RESTful Principles. Berdasarkan evaluasi kecocokan antara NFR, implementasi, dan kaidah RESTful Principles, ditemukan bahwa RESTful Principles cocok untuk diimplementasikan ke dalam Web Pencegahan Kecurangan. Baik dari sisi NFR maupun dari sisi kaidah RESTful Perinciples semua terpenuhi dengan sesuai dari implementasi Web Pencegahan Kecurangan ini.

Kata kunci : FDP, RESTful Principles, kecurangan

Abstract

Fraud Deterrence Propeller (FDP) is a good solution to deter fraud in an entity. Its implementation in the form of assessments also makes it easy to be carried out by small to large entities. This good solution can still be improved by digitizing the FDP. One way to digitize the FDP is to create a Fraud Deterrence Web that implements the FDP. To build a good fraud deterrence web, the right implementation rules are needed. One of the right implementation rules is the RESTful Principles rule. Based on the evaluation of the compatibility between NFR, implementation, and RESTful Principles, it is found that RESTful Principles are suitable for implementation into the Fraud Deterrence Web. Both in terms of NFRs and in terms of RESTful Perinciples rules are all met accordingly from the implementation of this Fraud Prevention Web.

Keywords: FDP, RESTful Principles, fraud

1. Pendahuluan

1.1 Latar Belakang

Fraud Deterrence Propeller (FDP) adalah kumpulan konsep pencegahan fraud yang meliputi enam konsep. Konsep-konsep tersebut di antaranya: *due diligent, enhancement, truthfulness and respect, efficacy of mind, reinforcement and communication,* dan *enforcement actions*. Masing-masing konsep memiliki indikatornya tersendiri untuk mengukur besar kemungkinan risiko terjadinya *fraud* [1].

FDP didesain untuk mengukur risiko terjadinya *fraud* pada sebuah entitas. Pada konteks ini, entitas bisa berupa perusahaan, organisasi, atau lembaga apapun yang memungkinkan untuk terjadi *fraud* di dalamnya. Implenetasi FDP bisa dilakukan dengan melakukan asesmen berdasarkan enam konsep yang ada pada FDP. Setiap konsep memiliki kuesioner tersendiri sesuai dengan cakupan konsep tersebut. Sebagai contoh, salah satu pertanyaan dari kuesioner untuk konsep *due diligent* adalah "Apakah terdapat akuntan internal?". Pertanyaan tersebut bertujuan untuk mengetahui apakah sebuah entitas memiliki akuntan internal. Jika sebuah entitas tidak memilki akuntan internal, maka risiko terjadinya *fraud* pada entitas tersebut dianggap meningkat.

Sampai artikel ini dibuat, FDP baru bisa diimplementasikan dengan cara mengisi kuesioner secara konvensional baik menggunakan kertas, *microsoft excel*, maupun *google form*. Dengan pengisian kuesioner dengan cara tersebut, hasil kuesioner tidak bisa diperiksa secara langsung oleh *reviewer*. Selain kuesioner yang tidak bisa langsung diperiksa oleh *reviewer*, bukti dari masing-masing poin pada kuesioner perlu dikumpulkan juga pada tempat tersendiri supaya tidak hilang, tertukar, atau teracak.Setelah asesmen diperiksa dan diverifikasi oleh *reviewer*, hasil penilaian juga perlu dicatat untuk mengetahui potensi *fraud* secara berkala.

Solusi yang bisa diimplementasikan untuk mengatasi kekurangan tersebut adalah dengan mendigitalisasi FDP tersebut. Dengan digitalisasi FDP, maka kekurangan teknis seperti kerapian dokumen, kerapian pencatatan, keaslian data, kepastian data *real time*, bisa ditangani. Selain itu, informasi mengenai tanggal asesmen dan identitas asesor bisa tersimpan dengan rapi pada *database* sistem tersebut.

Salah satu cara untuk mengimplementasikan solusi tersebut adalah dengan membuat web yang bisa mengimplementasikan FDP tersebut beserta fitur penunjang lain seperti pengaturan hak akses dan pencatatan log aktivitas pengguna web tersebut. Jika cara tersebut diimplementasikan, maka asesmen FDP bisa dilakukan dengan

tambahan beberapa kelebihan. Kelebihan tersebut di antaranya: perlindungan hak akses terhadap semua dokumen asesmen, merapikan penyimpanan data asesmen beserta bukti pada setiap poinnya, kecepatan proses dari mulai pelaksanaan asesmen hingga proses *review* asesmen tersebut, dan pencatatan log aktivitas pengguna pada aplikasi yang memungkinkan untuk melacak risiko *fraud* pada entitas dari waktu ke waktu.

Ketika *requirement engineering* selesai dilakukan, kebutuhan fungsional atau *functional requirements* (FR) yang terkumpul bisa dilanjutkan ke tahap *coding*. Tentu FR yang ada harus diimplementasikan menggunakan design pattern yang tepat dengan kebutuhan non fungsional atau *non functional requirements* (NFR) yang ada. Pada dokumen Spesifikasi Kebutuhan Perangkat Lunak (SKPL) aplikasi ini terdapat enam poin unik NFR. NFR tersebut adalah:

- *Performance*
- *Security*
- *Usability*
- *Maintainability*
- *Compatibility*
- *Availability*

Empat dari enam poin NFR yang ada pada dokumen tersebut—*performance, maintainability, compatibility,* dan *security*—merupakan NFR yang berkaitan erat dengan sisi *back end* (BE) web pencegahan *fraud*. Justfiikasi relasi antara NFR dengan sisi BE adalah sebagai berikut:

- *Performance*, sisi BE harus mampu menyediakan *server* yang mampu melayani *request* dengan cepat walaupun lalu lintas sedang tinggi.
- *Maintainability*, sisi BE harus mampu membuat web yang *source code* nya mudah dipahami supaya mudah dirawat atau diperbaiki ketika ditemukan *error.*
- *Compatibility*, sisi BE harus menyediakan *endpoint* dengan *interface* yang mudah digunakan dengan sistem lainnya. Selain itu *interface* harus kompatibel juga dengan standar protokol umum *server* seperti protokol HTTP dan struktur JavaScript Object Notation (JSON)
- *Security*, sisi BE harus mampu menangani autentikasi dan autorisasi untuk menjamin keamanan sistem dan data pengguna. Maka dari itu, implementasi FR web tersebut harus sangat memerhatikan NFR tersebut dari sisi BE.
- *Availability*, sisi BE harus memastikan web bisa diakses kapan saja. Dengan kata lain BE harus memastikan *up time* sebuah web memenuhi standar produksi sesuai kebutuhan web.

Diperlukan strategi yang tempat untuk mengimplementasikan FR tersebut agar hasilnya sesuai dengan NFR yang ada.

Strategi yang tepat untuk memenuhi semua kebutuhan tersebut adalah dengan mengimplementasikan *design pattern* yang tepat [2]. *Design pattern* yang mampu memenuhi NFR tersebut adalah RESTful API yang merujuk pada RESTful Principles. RESTful Principles adalah salah satu *design pattern* yang umum digunakan untuk pengembangan server dari sisi BE pada saat ini [3]. RESTful ini berakar dari arsitektur Representational State Transfer (REST). Sifat-sifat RESTful Principles yang mendukung NFR web FDP ini adalah *statelesness*, *client-server*, *uniform interface*, dan *cacheability*. Justifikasi antara sifat RESTful Principles dengan NFR yang tersedia adalah:

- *Statelesness* mendukung NFR *performance* dengan efisiensi penanganan *request*nya
- *Client-server* mendukung NFR *compatibility* karena ketidakterikatan antara *server* dengan semua *client* yang mengakses
- *Uniform interface* mendukung *usability* karena keseragaman *interface* yang mempermudah semua *client* yang ingin membuat *request* dengan konsistensi *interface*nya
- *Cacheability* mendukung *performance* dengan memungkinkannya sistem *caching* untuk meningkatkan performa server secara umum [4]

Poin *availability* secara umum dapat dipenuhi oleh RESTful Principles karena kebutuhan *availability* cenderung dipenuhi dari sisi infrastruktur. Dengan deskripsi penggunaan *design pattern* RESTful Principles tersebut, dapat diasumsikan RESTful Principles cocok untuk digunakan untuk mengimplementasikan rancangan solusi yang ada. Ketika rancangan solusi diimplementasikan dengan tepat, masalah yang melatarbelakanginya akan berhasil diselesaikan dengan tepat. Maka dari itu, risiko *fraud* yang dicegah menggunakan FDP beserta semua kelebihan dari FDP digital yang sudah dideskripsikan di awal diasumsikan bisa terpenuhi dengan tepat.

Berdasarkan publikasi berjudul “2023 State of the API Report” dari Postman, ada opsi arsitektur web selain REST atau RESTful Principles. Opsi arsitektur yang cukup populer untuk dipilih adalah arsitektur Simple Object Access Protocol (SOAP). SOAP adalah salah satu arsitektur yang sempat populer dalam pengembangan sebuah servis web. Perbedaan signifikan antara servis web yang menerapkan RESTful Principles dibanding dengan yang menerapkan SOAP adalah pada format pesan dan standar keamanan. Servis web yang menerapkan RESTful Principles umumnya menggunakan format pesan yang lebih ringan dengan menggunakan JSON. Kemanan servis web tersebut juga sifatnya tidak baku. Untuk servis web yang menggunakan SOAP, format pesan yang digunakan adalah XML yang umumnya lebih panjang dibanding JSON. Selain itu, keamanan pada server SOAP sifatnya baku dan kompleks. Maka dari itu, arsitektur SOAP akan lebih cocok untuk digunakan pada lingkungan korporasi yang sudah matang. Dengan demikian RESTful Principles tetap dipilih untuk menjadi arsitektur *back end* pada Web Pencegahan Kecurangan karena kebutuhan untuk memenuhi NFR pada poin *performance*. Selain itu, servis yang menerapkan RESTful Principles lebih ringan, mudah digunakan, *self descriptive*, cepat, mendukung banyak tipe data, dan menggunakan *bandwidth* yang lebih sedikit [5].

1.2 Topik dan Batasannya

Beberapa batasan pada penelitian ini adalah:

1. Pengembangan web mengacu pada draf dokumen SKPL yang dirancang oleh System Analyst
2. Library yang digunakan adalah FastAPI dalam bahasa Python
3. RDBMS yang digunakan adalah MySQL
4. API didokumentasikan menggunakan Swagger
5. Server API dideploy di VPS dengan OS Ubuntu
6. Provider VPS yang digunakan adalah BizNet Gio
7. Produk VPS yang dipilih adalah BizNet NeoLite
8. Pengembangan web dilakukan dalam waktu 6 bulan
9. Penelitian ini berfokus kepada pengembangan dari sisi *back end*

1.3 Tujuan

Beberapa tujuan dari pelaksanaan penelitian ini adalah:

1. Mengembangkan API dari Web Pencegahan Kecurangan dengan mengimplementasikan RESTful Principles
2. Menguji kecocokan antara implementasi REST API dengan kaidah RESTful Principles

2. Penjelasan Teoritis

Beberapa poin penjelasan pada sub bab di bab ini adalah terkait teori pendukung dalam proses penelitian ini.

2.1 RESTful Principles

RESTful Principles adalah prinsip atau kaidah yang diciptakan untuk memastikan bahwa sebuah *web service* memiliki skalabilitas yang baik, bersifat *stateless*, dan mudah dipelihara. RESTful Principles umumnya diimplementasikan pada pengembangan API sehingga API tersebut bisa diesbut sebagai RESTful API. RESTful Principles terdiri dari beberapa kaidah. Berikut adalah penjelasan dari kaidah-kaidahtersebut.

2.1.1 Uniform Interface

Uniform interface adalah kaidah RESTful Principles yang membahas tentang bagaimana API yang mengeimplementasikan RESTful Principles harus seragam. Seragam pada konteks ini adalah memiliki kesamaan pola pada penamaan *endpoint*. Umumnya *endpoint* menggunakan kata benda selagi dimungkinkan. Contoh bentuk *uniform interface* adalah:

- GET /buku yang berfungsi untuk mengambil data buku
- POST /buku yang berfunsi untuk memasukkan data buku
- DELETE /buku yang berfungsi untuk memasukkan data buku

2.1.2 Client-Server

Kaidah *client-server* adalah kaidah yang cenderung terpenuhi secara otomatis ketika membangun API yang mengimplementasikan RESTful Principles (RESTful API). Pada umumnya RESTful API dikembangkan sebagai *back end* yang terpisah dari *front end* pada sebuah web. Dengan diterapkannya *client-server* ini maka RESTful API sebagai *server* bisa melayani banyak *client* sekaligus.

2.1.3 Stateless

RESTful API harus bersifat *stateless*. Artinya, sebuah *server* tidak perlu menyimpan informasi konteks dari sebuah *request* yang pernah dilakukan oleh *client*. Maka dari itu, semua informasi yang dibutuhkan dari sebuah proses harus disertakan sepenuhnya oleh *client* dalam sebuah *request*. Jika diperlukan adanya *session*, maka *client* lah yang menyimpan informasi tersebut di *session* dari *client* itu sendiri.

2.1.4 Cacheable

Cacheable pada konteks ini adalah melakukan *caching* terhadap *response* dari *server*. Sebuah respon dari RESTful API harus bisa di cache milik *client*. Umumnya sebuah *response* bisa disimpan di *cache* meskipun ada baiknya RESTful API melabeli secara eksplisit bahwa sebuah *response* boleh disimpan di cache.

2.1.5 Layered System

Sebuah API dapat dikatakan mengimplementasikan *layered system* ketika di dalam servisnya API tersebut memiliki beberapa lapis aplikasi. Sebuah RESTful API sederhana bisa hanya memiliki *infrastructure layer* yang berkaitan dengan infrastruktur dan penyimpanan memori, *data access layer* yang berkaitan dengan penyimpanan data, dan *application layer* yang berkaitan dengan *business logic* dari sebuah servis.

2.2 API

Application Programming Interface (API) adalah istilah yang secara umum berarti antarmuka dari sebuah aplikasi namun dalam konteks pemrograman. API biasanya berbentuk kumpulan kode siap pakai yang digunakan untuk mengakses sebuah aplikasi dari aplikasi lainnya. Beberapa contoh API adalah GraphQL API, SOAP API, dan RESTful API.

2.3 RESTful API

RESTful API adalah API yang dibangun dengan mengimplementasikan RESTful Principles. RESTful API sifatnya sudah lebih mengerucut ke arah *web service* yang menerapkan prinsip RESTful dan menggunakan protokol HTTP. *Web service* yang modern umumnya menggunakan RESTful API karena sifatnya yang mudah diakses secara luas.

2.4 VPS

Virtual Private Server (VPS) adalah *server* virtual yang umumnya disewakan kepada pengguna sehingga para pengguna memiliki kontrol untuk melakukan konfigurasi *server* secara mandiri. VPS umumnya dipilih karena lebih efisien dari sisi biaya karena pengguna tidak perlu membangun *server* sendiri. Selain itu keamanan *server* lebih mudah dikelola karena umumnya sudah diatur oleh penyedia VPS. Sehingga, pengguna tidak perlu melakukan konfigurasi infrastruktur dan mengurus keamanannya secara mandiri.

2.5 Docker

Docker berfungsi untuk mempermudah *developer* untuk melakukan *deployment* aplikasi. Kemudahan tersebut tercapai karena Docker melakukan *containerization* pada sebuah *environment* [6]. Dengan *containerization* tersebut proses *deployment* tidak lagi mengharuskan *developer* untuk melakukan konfigurasi ulang pada VPS

2.6 RDBMS

Relational Data Base Management System (RDBMS) adalah sistem manajemen *database* yang bersifat relasional. Arsitektur relasional umumnya dipilih untuk *database* transaksional atau *database* yang memiliki banyak aktivitas *read and write*. Selain itu *database* relasional juga dipilih ketika banyak keterkaitan antar tabel. *Database* relasional lebih dipilih ketika banyak keterkaitan antar tabel karena secara alami memiliki sistem *primary key* dan *foreign key* yang memudahkan *query* yang melibatkan banyak tabel.

2.7 Financial Fraud

Financial Fraud atau kecurangan adalah aktivitas ilegal dengan tujuan mencari keuntungan sebesar-besarnya untuk diri sendiri. Kecurangan ini bisa dilakukan dengan berbagai cara, contohnya adalah penggelapan aset dan manipulasi laporan keuangan [1].

2.8 Financial Statement Fraud

Financial Statement Fraud atau manipulasi laporan keuangan adalah salah satu bentuk kecurangan yang mungkin terjadi pada sebuah entitas. Pelaku manipulasi laporan keuangan bisa mendapatkan keuntungna secara langsung dari hal ini. Contoh celah keuntungannya adalah memanipulasi anggaran yang dinaikkan supaya kelebihan anggaran tersebut bisa masuk ke rekening pribadi pelaku.

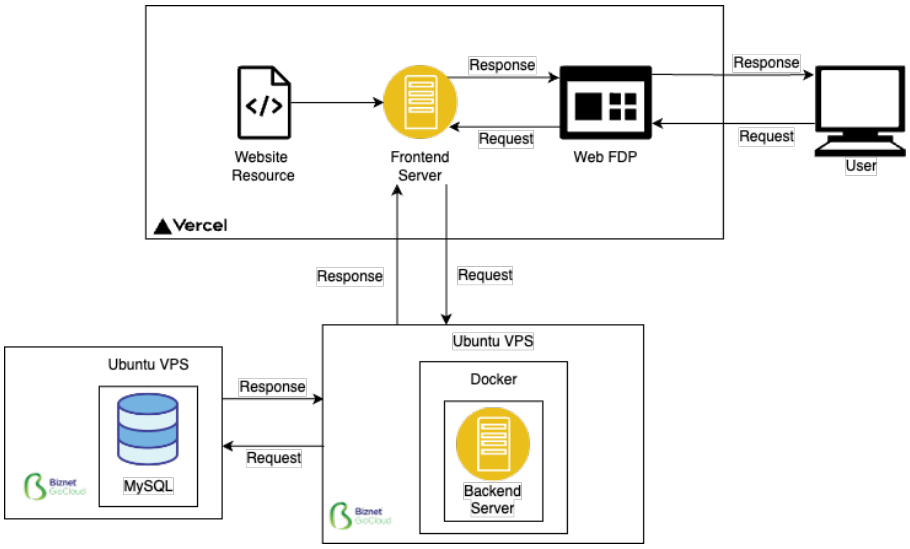
2.9 FDP

Fraud Deterrence Propeller (FDP) adalah sistem pencegahan kecurangan dengan cara mengases sebuah entitas dan memberitahu entitas tersebut akan potensi terjadinya kecurangan. Sistem ini diimplementasikan dengan bentuk asesmen yang dilakukan oleh admin entitas dan akan ditinjau oleh *reviewer* internal dan *reviewer* eksternal [1].

3. Sistem yang Dibangun

3.1 Rancangan

Berdasarkan dokumen SKPL Web Pencegahan Kecurangan ini, ada FR dan NFR yang menjadi acuan ketika merancang web ini. Dengan mempertimbangkan FR dan NFR tersebut, Gambar 1 menunjukkan hasil rancangan arsitektur Web Pencegahan Kecurangan.



Gambar 1 Arsitektur

Deployment dari BE Web Pencegahan Kecurangan akan dilakukan di sebuah VPS. *Deployment* BE dilakukan di satu VPS karena arsitekturnya yang menerapkan *monolithic* sehingga tidak memerlukan banyak VPS atau *server*. Arsitektur *monolithic* dipilih karena dengan mempertimbangkan FR Web Pencegahan Kecurangan. Berdasarkan FR tersebut, ditemukan bahwa *business process* yang ada pada Web Pencegahan Kecurangan ada dua. *Business process* tersebut adalah pencegahan kecurangan dengan asesmen dan pendeteksi kecurangan dengan perhitungan rumus pendeteksi kecurangan. Kedua *business process* tersebut tidak memerlukan *multi-cloud server* [7], integrasi IoT [8], maupun prosedur operasional skala besar [9]. Dengan kata lain, arsitektur *monolithic* dipilih karena BE Web Pencegahan Kecurangan belum memerlukan skalabilitas sebesar itu. Hal ini sesuai dengan NFR pada SKPL Web Pencegahan Kecurangan yang tidak mencantumkan *scalability* pada kualitas atributnya.

Untuk mempermudah *deployment* BE Web Pencegahan Kecurangan pada VPS, peneliti memilih VPS dengan *Operating System* (OS) Ubuntu. Selain Ubuntu, OS yang biasa digunakan di *web server* adalah Debian. Namun, Ubuntu pada umumnya dinilai lebih mudah digunakan dibanding Debian [10]. Salah satunya adalah karena proses instalasi di Ubuntu lebih efisien dibanding Debian. Sedangkan pada Debian instalasi lebih mungkin untuk dikustomisasi namun lebih sulit digunakan. Pada BE Web Pencegahan Kecurangan, kemudahan penggunaan lebih diutamakan dibanding sifat *customizable*. Kemudahan penggunaan OS pada VPS dipilih karena mempertimbangkan NFR *maintainability* pada SKPL. Mudahnya penggunaan OS akan memudahkan juga pada pemeliharaan sistem dan pembaharuan kode pada *server*.

Selain dengan memilih OS Ubuntu, *maintainability* juga didukung dengan penggunaan Docker. Dengan digunakannya Docker maka *developer* tidak perlu melakukan konfigurasi ulang VPS. Selain itu, Docker juga memastikan seluruh *environment* konsisten dari *development environment* sampai *production environment* [11]. *Developer* hanya perlu menginstal Docker di VPS dan menyediakan Dockerfile atau dokumen yml yang digunakan pada *development environment*. Dokumen tersebut berfungsi untuk membuat *image* yang sama supaya servis BE pada *production environment* bisa sama seperti pada *development environment*.

Berbeda dengan VPS untuk BE, VPS untuk *database* tidak menggunakan Docker dengan pertimbangan NFR dari kualitas atribut *performance*. *Database* MySQL akan berjalan dengan performa lebih baik ketika dijalankan langsung pada VPS Ubuntu tanpa Docker karena akan bersifat *native* [12]. Artinya MySQL akan berjalan langsung di atas OS tanpa melalui lapisan *containerization* dari Docker. Menjalankan MySQL secara *native* juga dipilih karena MySQL dijalankan pada sebuah VPS yang terpisah dari servis BE. Maka dari itu, MySQL tidak perlu diisolasi kembali dengan *containerization* dari Docker.

Pemisahan VPS dan *containerization* pada keseluruhan servis BE Web Pencegahan Kecurangan juga mendukung NFR pada poin *security*. VPS yang terpisah akan mencegah dampak ketika terjadi serangan siber. Jika *server* BE terkena serangan atau *breach*, maka *database server* tidak akan secara otomatis terserang. Selain itu, *containerization* juga memungkinkan diberlakukannya pembatasan CPU dalam hal penggunaan memori. Maka dari itu, ketika terjadi serangan Distributed Denial of Service (DDoS), VPS atau *server* BE dari Web Pencegahan Kecurangan tidak akan langsung *down*.

Pemenuhan NFR dari atribut *security* tersebut berdampak positif terhadap NFR pada atribut *availability*. Dengan *server* yang lebih aman, maka ketersediaan layanan Web Pencegahan Kecurangan akan lebih tinggi juga. Ketika terjadi serangan siber terjadi, ada kemungkinan layanan Web Pencegahan Kecurangan akan terganggu. Contoh serangan siber yang dapat mengganggu layanan web ini adalah DDoS. Serangan DDoS akan menyebabkan *server* melambat bahkan mati karena penggunaan CPU yang sudah mencapai batas [13]. Permasalahan pada *server* tersebut tentu berdampak negatif juga pada atribut *availability* dari Web Pencegahan Kecurangan. Jika atribut *security* sudah bisa terjaga dan terpenuhi, maka atribut *availability* akan lebih terpenuhi.

Terdapat cara lain untuk menjamin atribut kualitas *availability* selain dengan menjamin keterpenuhan atribut *security*. Cara lain untuk menjamin atribut *availability* adalah dengan memastikan kualitas infrastruktur tempat *deployment* sebuah servis terjamin. BizNet Gio sebagai penyedia VPS yang dipilih untuk Web Pencegahan Kecurangan telah memenuhi standar ISO 27001 dan ISO 27017. Salah satu contoh standar yang harus dipenuhi dari ISO 27017 adalah keamanan data pada layanan *cloud*. Standar tersebut menunjukkan bahwa BizNet Gio sudah memenuhi standar *security* dan *availability* milik ISO [14].

Setelah perancangan arsitektur sudah sesuai dengan FR dan NFR, ada beberapa hal yang perlu dipersiapkan sebelum *server* BE bisa dijalankan di VPS. Hal tersebut adalah VPS itu sendiri dan *database* pada VPS untuk *database*.

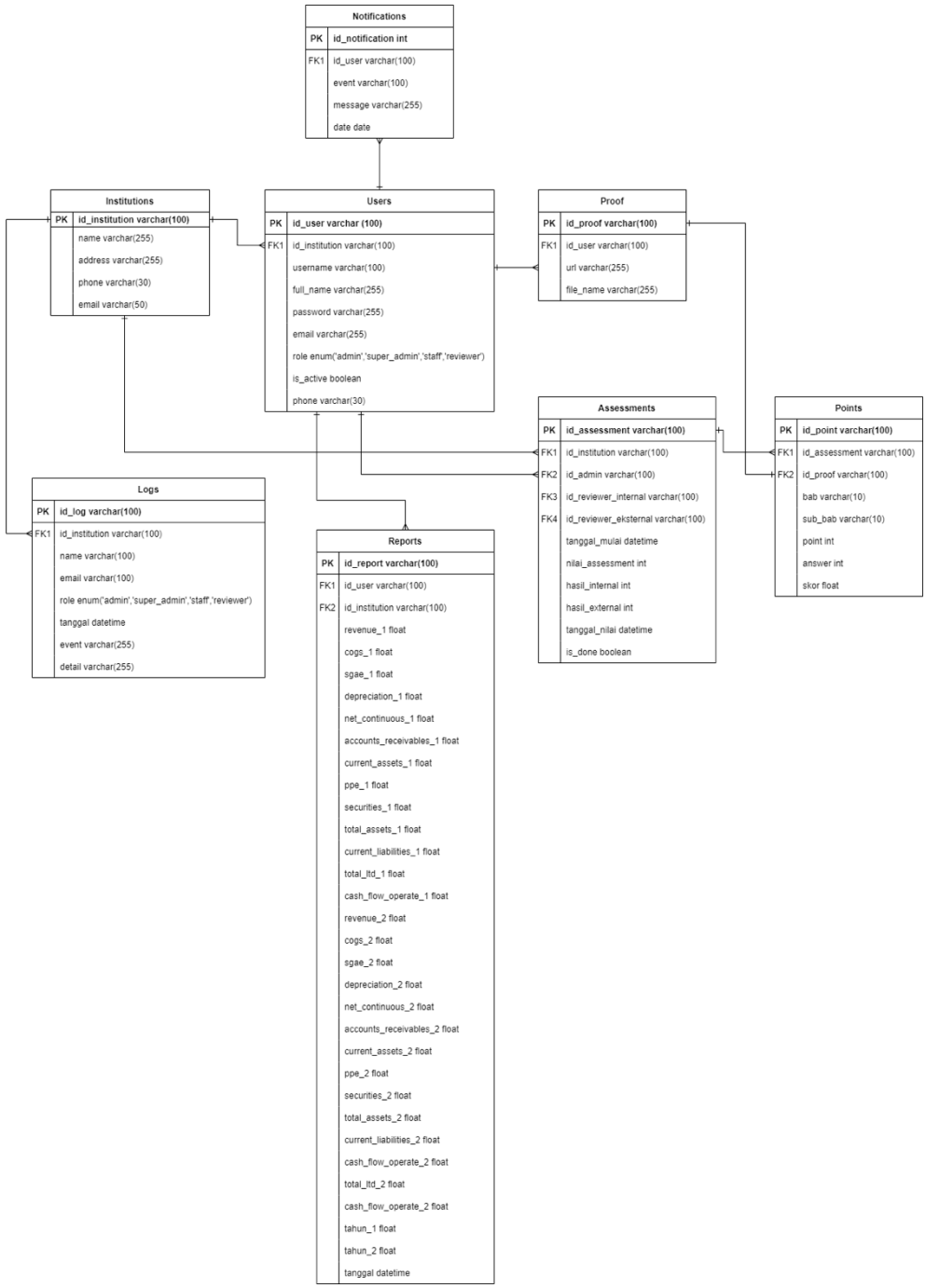
3.1.1 VPS

VPS berperan sebagai infrastruktur tempat berjalannya *database server* dan *backend server*. Sebelum bisa digunakan untuk menjalankan kedua server tersebut, VPS perlu dikonfigurasi menyesuaikan kebutuhan server. VPS yang digunakan pada Web Pencegahan Keuangan adalah VPS dengan OS Ubuntu. Web Pencegahan Keuangan menggunakan dua VPS berbeda untuk menjalankan *backend* dan *database*. Penggunaan dua database berbeda tersebut bertujuan untuk menghemat biaya VPS. Menyewa dua VPS kapasitas kecil lebih murah dibanding menyewa satu VPS kapasitas besar.

Proses konfigurasi VPS *database* dilakukan dengan menginstall MySQL dan mengonfigurasi nginx VPS tersebut supaya database bisa diakses dari internet. Selain instalasi hal tersebut, perlu dikonfigurasi juga sertifikat SSL supaya koneksi yang dibangun bisa lebih aman. Koneksi dengan sertifikat SSL tersedia bisa lebih aman karena protokol yang digunakan sudah HTTPS. Sedangkan konfigurasi VPS *backend* dilakukan dengan instalasi docker untuk memudahkan *deployment* kode *backend*. Konfigurasi sertifikat SSL dilakukan pada kedua VPS untuk memastikan keamanan sistem dan meningkatkan integritas sistem.

3.1.2 Database

Setelah VPS dikonfigurasi dan MySQL telah diinstal pada salah satu VPS, *database* bisa disiapkan untuk menjalankan sistem. Proses menyiapkan *database* dimulai dari implementasi rancangan *database* berdasarkan *Entity Relationship Diagram* (ERD) yang tersedia menjadi sebuah *database*. Setelah *database* sudah dibuat, hak akses pengguna perlu dikonfigurasi supaya *database* lebih aman dan tidak bisa diakses oleh sembarang orang. Setelah hak akses pengguna sudah diatur, *datbase* sudah bisa diakses dengan aman oleh *backend*.



Gambar 2 *Entity Relationship Diagram*

ERD pada Gambar 2 adalah ERD yang akan diimplementasikan menjadi *database* dari Web Pencegahan Kecurangan. ERD tersebut sudah mencakup dua *business process* utama web ini. Asesmen yang dilakukan dari proses pencegahan kecurangan akan disimpan pada tabel Assessments beserta dengan informasi siapa yang mengisi asesmen, siapa yang melakukan *review* terhadap asesmen tersebut, dan kapan asesmen tersebut dilakukan. Sedangkan untuk proses pendeteksian kecurangan data utama proses tersebut disimpan pada tabel Reports yang menyimpan informasi penting dari laporan keuangan untuk mendeteksi potensi kecurangan.

4. Implementasi

Implementasi dilakukan dengan mengacu pada SKPL baik dari sisi fungsional maupun non fungsional. Secara fungsional implementasi akan mengacu pada FR yang berisi semua kebutuhan dari Web Pencegahan Kecurangan. Dari sisi non fungsional implementasi mengacu pada NFR beserta diagram arsitektur yang ada pada Gambar 1. Diagram pada Gambar 1 dapat ditemukan juga dalam SKPL. Implementasi non fungsional lebih banyak diimplementasikan di bagian pengambilan keputusan teknis mengenai *tech stack*, arsitektur, dan infrastruktur.

4.1 Definisi Endpoint

Implementasi dari FR lebih banyak diwujudkan dalam bentuk *endpoint* yang ada pada sebuah RESTful API. Setiap FR yang ada perlu ditangani dengan menyediakan *endpoint* untuk melayani semua *business process* di baliknya. Sebuah *endpoint* bisa menangani lebih dari satu *business process* dan sebuah *business process* bisa menggunakan lebih dari satu *endpoint*. Sebagian besar logika *business process* juga terjadi pada sebuah *endpoint*.

Endpoint dalam konteks ini adalah sebuah URL yang dapat diakses oleh *client*. Biasanya sebuah RESTful API terdiri atas banyak *endpoint* yang memiliki fungsi tersendiri. Sebuah *endpoint* yang dibangun menggunakan *library* FastAPI memiliki anatomi seperti pada Gambar 3

```python
@router.get( path: '/report', tags=['Detection - Admin'])
async def get_report_list(user: UserDB = Depends(get_user)) → JSONResponse:
    if user.role not in ['admin', 'staff']:
        return create_response(
            message="Forbidden access",
            success=False,
            status_code=status.HTTP_403_FORBIDDEN
        )

    reports = get_report_beneish()
    for report in reports:
        report['tanggal'] = report['tanggal'].strftime('%Y-%m-%d %H:%M:%S')

    return create_response(
        message="Fetch data success",
        success=True,
        status_code=status.HTTP_200_OK,
        data=reports
    )
```

Gambar 3 Endpoint

Struktur *endpoint* tersebut bisa dijabarkan sebagai berikut:

a. Path

Sebuah *endpoint* memiliki *path* atau URL yang menjadi alamat ketika akan melakukan request. Pada *endpoint* yang terdapat pada Gambar 3, *path* ditentukan dengan sintaks "@router.get('/report', tags=['Detection – Admin'])". Pada sintaks tersebut yang menentukan *path* adalah '/report'. *Endpoint* yang dihasilkan adalah '/report'

b. Method dan parameter

Method pada sebuah *endpoint* sifatnya tidak baku dalam hal penamaan. Nama sebuah *method* di FastAPI dibebaskan selama mengikuti aturan penamaan *method* bahasa python. Untuk parameter dari sebuah *endpoint* letaknya ada di tempat yang sama dengan parameter sebuah *method*. Parameter biasanya berisi hal yang diperlukan dari sebuah *request body* atau *header* yang berisikan *auth token*.

c. Logic

Setelah menentukan *path*, *method*, dan parameter, tahap selanjutnya adalah mengembangkan *programming logic* sesuai dengan FR yang akan dipenuhi. Pada *endpoint* yang terdapat pada Gambar 3 *programming logic* yang terjadi adalah pengecekan *role* sebelum mengambil data laporan deteksi kecurangan di *database*. Setelah data laporan diambil, dilakukan pengubahan format tanggal dari tipe datetime menjadi tipe string sebelum dikembalikan sebagai *response body*.

4.2 Pengembangan Endpoint

Pengembangan *endpoint* adalah aktivitas utama dalam memenuhi FR dari sisi *back end*. Sebagian besar *business process* akan dipenuhi dengan implementasi membuat *endpoint* dengan *programming logic* yang sesuai dengan kebutuhan. Maka dari itu, perlu dilakukan pengembangan *endpoint* secara detail untuk setiap FR. Pada sub bab selanjutnya terdapat informasi pemenuhan kebutuhan pada FR beserta URL *endpoint* sebagai implementasinya. FR yang tidak disertakan pada Tabel 1, Tabel 2, Tabel 3, Tabel 4, dan Tabel 5 adalah FR yang implementasinya cenderung dipenuhi dari sisi *front end* atau FR yang ditangguhkan karena keterbatasan waktu.

4.2.1 Endpoint Superadmin

Superadmin adalah *role* khusus sehingga tidak memiliki FR registerasi. Akun yang dimiliki oleh superadmin dibuatkan langsung oleh developer.

Tabel 1 FR Superadmin

FR ID	Kebutuhan Fungsional	Endpoint URL	Tingkat Kepentingan
FR-SUP-01	Superadmin harus melakukan login di halaman login untuk mendapatkan hak akses	[POST] /api/auth	*High*
FR-SUP-03	Superadmin dapat melihat list akun Admin yang menunggu konfirmasi pasca registrasi	[GET] /api/admin	*High*
FR-SUP-04	Superadmin dapat menerima akun Admin yang mendaftar	[POST] /api/confirm	*High*
FR-SUP-05	Superadmin dapat menolak akun Admin yang mendaftar	[DELETE] /api/reject	*High*
FR-SUP-06	Superadmin dapat melihat list akun Admin yang terdaftar	[GET] /api/admin	*High*
FR-SUP-07	Superadmin dapat mengaktifkan akun Admin yang terdaftar pada aplikasi	[GET] /api/verify/{id}	*High*
FR-SUP-08	Superadmin dapat menonaktifkan akun Admin yang terdaftar pada aplikasi	[GET] /api/verify/{id}	*High*

4.2.2 Endpoint Admin

Akun dengan *role* admin terkait secara langsung dengan data entitas. Ketika seseorang ingin mendaftarkan dirinya sendiri sebagai admin maka dia harus mendaftarkan juga entitas miliknya. Sebuah entitas hanya bisa memiliki satu akun admin. Tetapi, sebuah entitas bisa memiliki banyak staff dan *reviewer* internal.

Tabel 2 FR Admin

FR ID	Kebutuhan Fungsional	Endpoint URL	Tingkat Kepentingan
FR-ADM-01	Admin dapat melakukan registrasi di halaman registrasi	[POST] /api/register	*High*
FR-ADM-02	Admin harus melakukan login di halaman login untuk mendapatkan hak akses	[POST] /api/auth	*High*
FR-ADM-05	Admin dapat menambahkan akun Staf dan Reviewer Internal	[POST] /api/account	*High*
FR-ADM-06	Admin dapat melihat list akun Staf dan Reviewer Internal yang ditambahkan	[GET] /api/staff	*High*
FR-ADM-07	Admin dapat mengaktifkan akun Staf dan Reviewer Internal yang ditambahkan	[PATCH] /api/alter	*High*
FR-ADM-08	Admin dapat menonaktifkan akun Staf dan Reviewer Internal yang ditambahkan	[PATCH] /api/alter	*High*
FR-ADM-09	Admin dapat mengisi *Fraud Assessment*	[POST] /api/assessment [POST] /api/point	*High*

FR-ADM-10	Admin dapat mengubah isian *Fraud Assessment* selama *assessment* belum disubmit	[PATCH] /api/point	*High*
FR-ADM-11	Admin dapat melihat list riwayat *Fraud Assessment*	[GET] /api/assessments	*High*
FR-ADM-12	Admin dapat melihat detail nilai *Fraud Assessment* yang diberikan oleh Reviewer Internal dan Reviewer Eksternal	[GET] /api/assessment/{id}	*High*
FR-ADM-14	Admin dapat melakukan *Fraud Detection*	[POST] /report	High
FR-ADM-15	Admin dapat melihat list riwayat penilaian *Fraud Detection*	[GET] /api/report	*High*
FR-ADM-16	Admin dapat melihat detail nilai hasil *Fraud Detection*	[POST] /api/report/{id}	*High*
FR-ADM-18	Admin dapat melihat list riwayat aktifitas *user* yang dilakukan selama mengakses aplikasi	[GET] /api/log	*Medium*

4.2.3 Endpoint Staff

Seseorang dengan *role* staff dapat mengakses web dan melakukan *login*. Tetapi, seorang staff tidak bisa melakukan registerasi untuk dirinya sendiri. Hanya admin sebuah entitas yang bisa mendaftarkan staff untuk entitas tersebut.

Tabel 3 FR Staff

FR ID	Kebutuhan Fungsional	Endpoint URL	Tingkat Kepentingan
FR-STF-01	Staf harus melakukan login di halaman login untuk mendapatkan hak akses	[POST] /api/auth	*High*
FR-STF-04	Staf dapat mengisi *Fraud Assessment*	[POST] /api/assessment [POST] /api/point	*High*
FR-STF-05	Staf dapat mengubah isian *Fraud Assessment* selama *assessment* belum disubmit	[PATCH] /api/point	*High*
FR-STF-06	Staf dapat melihat list riwayat *Fraud Assessment*	[GET] /api/assessments	*High*
FR-STF-07	Staf dapat melihat detail nilai *Fraud Assessment* yang diberikan oleh Reviewer Internal dan Reviewer Eksternal	[GET] /api/assessment/{id}	*High*
FR-STF-09	Staf dapat melakukan *Fraud Detection*	*[POST] /report*	*High*
FR-STF-10	Staf dapat melihat list riwayat penilaian *Fraud Detection*	[GET] /api/report	*High*
FR-STF-11	Staf dapat melihat detail nilai hasil *Fraud Detection*	[POST] /api/report/{id}	*High*
FR-STF-13	Staf dapat melihat list riwayat aktifitas *user* yang dilakukan selama mengakses aplikasi	[GET] /api/log	*Medium*

4.2.4 Endpoint Reviewer Internal

Cara kerja akun seorang *reviewer* internal mirip dengan staff pada bagian *login* dan registerasi. Keduanya dapat melakukan *login* namun tidak dapat melakukan registerasi secara mandiri. *Reviewer* internal masih terkait dengan sebuah entitas.

Tabel 4 FR Reviewer Internal

FR ID	Kebutuhan Fungsional	Endpoint URL	Tingkat Kepentingan
FR-RIN-01	Reviewer Internal harus melakukan login di halaman login untuk mendapatkan hak akses	[POST] /api/auth	*High*
FR-RIN-04	Reviewer Internal dapat melihat list *Fraud Assessment* yang belum dinilai olehnya	[GET] /api/assessments/list	*High*
FR-RIN-05	Reviewer Internal dapat melakukan penilaian pada *Fraud Assessment* yang belum dinilai olehnya	[POST] /api/assessments/evaluation	*High*
FR-RIN-06	Reviewer Internal dapat melihat list *Fraud Assessment* yang sudah dinilai olehnya	[GET] /api/assessment/{id}	*High*
FR-RIN-07	Reviewer Internal dapat melihat detail riwayat *Fraud Assessment* yang telah dinilai olehnya	[GET] /api/assessments/insight/{id}	*High*

4.2.5 Endpoint Reviewer External

Akun *reviewer* eksternal adalah satu-satunya jenis akun yang tidak terkait dengan entitas manapun. Akun *reviewer* eksternal seperti akun superadmin yang dibuatkan langsung oleh pengembang.

FR ID	Kebutuhan Fungsional	Endpoint URL	Tingkat Kepentingan
FR-REK-01	Reviewer Eksternal harus melakukan login di halaman login untuk mendapatkan hak akses	[POST] /api/auth	*High*
FR-REK-04	Reviewer Eksternal dapat melihat *Fraud Assessment* yang sudah dinilai oleh Reviewer Internal	[GET] /api/assessments/list	*High*
FR-REK-05	Reviewer Eksternal dapat melakukan penilaian pada *Fraud Assessment* yang belum dinilai olehnya, jika Reviewer Internal sudah menilai *assessment* tersebut	[POST] /api/assessments/evaluation	*High*
FR-REK-06	Reviewer Eksternal dapat melihat list *Fraud Assessment* yang sudah dinilai olehnya	[GET] /api/assessment/{id}	*High*
FR-REK-07	Reviewer Eksternal dapat melihat detail riwayat *Fraud Assessment* yang telah dinilai olehnya	[GET] /api/assessments/insight/{id}	*High*

4.3 Tech Stack

Pada Tabel 5 tercantum *tech stack* yang digunakan untuk mengembangkan semua *endpoint* yang disebutkan pada sub bab sebelumnya. Bukti implementasi dari *tech stack* yang digunakan terdapat pada bagian lampiran mulai dari Lampiran 1 sampai Lampiran 5

Tabel 5 *Tech Stack*

Jenis	*Tools*
VPS	BizNet NEO Lite
Drive	BizNet NEO Object Storage
OS	Ubuntu-22.04
RDBMS	MySQL
Programming Language	Python
Library	FastAPI
Versioning	Git (GitHub)

Documentation	Swagger
Containerization Platform	Docker

Implementasi RESTful Principles terjadi pada saat pembangunan keseluruhan servis BE Web Pencegahan Kecurangan. RESTful Principles diimplementasikan secara spesifik pada API dari servis BE web ini. Sehingga API web ini bisa disebut sebagai RESTful Principles. Bahasa Python dan *framework* FastAPI digunakan untuk mengimplementasikan RESTful API dengan pertimbangan *learning curve* dan batasan waktu pengembangan web. Dengan *learning curve* yang relatif mudah maka waktu pengembangan web bisa dilakukan dengan lebih cepat. Dari sisi performa Python tidak lebih cepat dari Golang untuk servis dengan layanan tinggi [15]. Tetapi, dengan menimbang teknologi infrastruktur yang tersedia saat ini, performa Python masih bisa diterima untuk sebuah produk web terutama untuk produk rintisan. Sehingga Python masih mampu memenuhi standar kebutuhan untuk implementasi RESTful API.

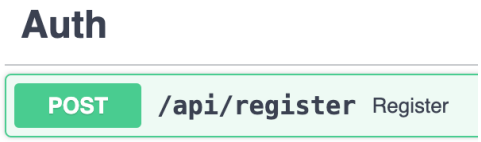
Selain bahasa Python itu sendiri, *library* pemrograman yang digunakan juga mendukung performa Python untuk bisa lebih efisien. FastAPI adalah *library* Python yang digunakan untuk membangun RESTful API dengan mudah dan asiknronus. Sifat asinkronus pada *library* ini semakin membantu performa Python di *environment production*. Kemudahan juga menjadi pertimbangan untuk pemilihan FastAPI sebagai *library* RESTful API yang dibangun. Kemudahan ini termasuk bagian dari *maintainability* yang ada pada NFR.

Pemilihan RESTful Principles untuk membangun API itu sendiri didasari pada kebutuhan NFR pada poin *compatibility* yang memungkinkan API yang dibangun untuk digunakan dengan mudah dari berbagai layanan yang berbeda jika satu saat aplikasi sudah lebih besar dan berkembang. Sifat RESTful pada poin *uniform interface* membuatnya lebih kompatibel karena sifat keseragaman pada *interface*, *request body*, dan *request response*. *Interface* yang menggunakan protokol HTTP standar seperti GET, POST, PUT, DELETE, dan PATCH membuat *client* akan lebih mudah untuk mengaksesnya. [16].

```
@router.post( path: '/register', tags=['Auth'])
async def register(data: RegisterForm) → JSONResponse:
```

Gambar 4 HTTP Post

Gambar 4 adalah contoh *endpoint* dengan HTTP *method* POST. URL yang ditangani oleh kode tersebut adalah '/register'. Kode "tags=['Auth']" berfungsi sebagai penanda bahwa *endpoint* tersebut berkaitan dengan fitur autentikasi. "Tag" tersebut akan berpengaruh pada dokumentasi API yang tampil pada Swagger. *Endpoint* berikut berfungsi untuk menangani *request* dari pengguna yang ingin melakukan registerasi akun admin dan entitas. Contoh tampilan *endpoint* tersebut pada Swagger adalah seperti pada Gambar 5 berikut.



Gambar 5 Swagger

Sedangkan pada Gambar 6 terdapat contoh *endpoint* yang menangani *request* dari *client* yang akan memerika status autentikasi mereka.

```
@router.get( path: "/auth", tags=['Auth'])
async def check(access_token: str = Depends(oauth2_scheme)) → JSONResponse:
```

Gambar 6 HTTP Get

Contoh *endpoint* tersebut juga menunjukkan sifat *stateless* yang menyebabkan RESTful API lebih ringan secara performa. Terlebih lagi respon dari setiap *request* tersebut bisa disimpan di *cache* dari sisi *client* sehingga beberapa informasi bisa disimpan tanpa harus membuat *request* berulang kali. Sifat *stateless* dan *cacheable* ini sangat berguna pada fitur autentikasi aplikasi yang membuat *server* tidak harus mengingat siapa *client* yang membuat *request*. *Server* hanya perlu memeriksa dari *authorization header* dari setiap *request* yang umumnya memuat *authorization token* sebagai penanda siapa yang melakukan *request*.

Selain pengembangan RESTful API, keseluruhan servis BE juga mencakup servis *database*. Servis *database* dibangun dan dikonfigurasi pada VPS tersendiri dan menghasilkan keluaran berupa Alamat IP *database* yang bisa diakses oleh RESTful API. Untuk mengamankan akses terhadap *database*, *developer* bisa mengatur hak akses pengguna yang dimiliki oleh RESTful API. Dengan cara tersebut maka tidak sembarang orang atau servis bisa mengakses *database*. VPS yang terpisah juga mampu mencegah dampak serangan siber menyebar ke seluruh system.

Setelah RESTful API selesai dikembangkan, hal yang tidak boleh terlewat adalah mengamankan *endpoint* dari RESTful API tersebut. Salah satu cara yang bisa dilakukan adalah menambahkan sertifikat SSL pada server BE supaya protokol yang digunakan bisa ditingkatkan dari HTTP menjadi HTTPS. Protokol HTTP dinilai kurang aman karena pengiriman informasi terjadi dalam bentuk *plain text* yang tidak terenkripsi [17]. Dengan adanya sertifikat SSL pada protokol HTTPS, pengiriman informasi menjadi lebih aman karena informasi dikirim dengan bentuk terenkripsi [18]. Penambahan sertifikal SSL ini diimpelementasikan menggunakan Traefik. Traefik bisa dipasang dengan menjalankan *container* Traefik pada Docker yang ada di dalam VPS *server* BE.

4.4 Evaluasi

Setelah implementasi selesai dilakukan, maka tahap evaluasi bisa dimulai. Karena FR dianggap telah terpenuhi dengan *endpoint* yang sudah ada, maka evaluasi akan dilaksanakan mengacu pada NFR.

4.4.1 Performance

Dari sisi performance, pemilihan RESTful dibanding SOAP sudah terbukti berdampak positif terhadap *performance* dari *back end* Web Pencegahan Kecurangan sebagaimana tercantum pada artikel yang membandingkan performa RESTful dibanding SOAP dengan judul “Web Services: A Comparison of Soap and Rest Services” [5] . Pemilihan library FastAPI dalam bahasa python pun sudah terbukti cukup untuk sebuah web rintisan. Maka dari itu aspek *performance* dapat dianggap terpenuhi oleh implementasi ini.

4.4.2 Maintainabilty

Aspek *maintainability* dipenuhi dengan cara pemilihan beberapa *tech stack* seperti Ubuntu untuk OS, python untuk bahasa pemrograman, dan Docker untuk *containerization platform*. Ubuntu dan python memiliki *learning curve* yang relatif lebih mudah dibanding OS dan bahasa pemgrograman lain. Dengan kemudahan tersebut, maka pengembangan web akan lebih mudah meskipun dilanjut oleh pengembang baru yang belum familiar dengan *environment* Web Pencegahan Kecurangan. Docker pun mempermudah proses *deployment* sehingga pengembang dengan peran DevOps tidak perlu melakukan *setup* ulang server.

4.4.3 Compatibility

Compatibility terpenuhi dari sisi *uniform interface* yang ada pada RESTful Principles. Dengan respon yang sudah diseragamkan maka *back end* akan lebih kompatibel untuk berbagai platform berbeda.

4.4.4 Security

Aspek *security* dapat dipenuhi dengan implementasi JSON Web Token (JWT) sebagai autentikator dari setiap *endpoint* yang dibuat. Setiap *endpoint* yang memerlukan hak akses tertentu harus menyertakan auth token pada *request headernya*. Selain itu tata cara pengelolaan VPS yang memisahkan antara *database server* dengan *back end server* juga memperkuat sisi keamanan ini. Selain itu, penyedia VPS tempat *deployment* untuk *back end server* dan *database server* telah memiliki sertifikasi ISO 27001 dan ISO 27017.

4.4.5 Availability

Dengan penyedia VPS yang memenuhi ISO 27001 dan ISO 27017 maka dapat diyakini bahwa VPS tersebut cukup terpercaya *availability* dari layanan VPS mereka. Selain itu, penyedia VPS juga mencantumkan pada halaman "About Us" bahwa mereka menjamin SLA hingga 99.9% [19]. Gambar 7 menunjukkan klaim yang dicantumkan oleh penyedia VPS.

Gambar 7 SLA 99.99%

5. Kesimpulan dan Saran

Terdapat beberapa kesimpulan dan saran terkait hasil penelitian pada artikel ini. Pada bagian kesimpulan akan dijelaskan inti dari hasil penelitian ini beserta dengan jawaban atas masalah yang telah dirumuskan pada bab pertama. Sedangkan pada bagian saran akakn dijelaskan rekomendasi untuk menindaklanjuti hasil penelitian ini.

5.1 Kesimpulan

Berdasarkan evaluasi tersebut, dapat disimpulkan RESTful Principles dapat diipmlementasikan dengan baik untuk Web Pencegahan Kecurangan. Dalam kata lain RESTful Principles cocok untuk web ini. Untuk pengujian kesesuaian antara implementasi RESTful API dengan FR secara *business process* akan dilakukan oleh Quality Assurance (QA) engineer dan tidak masuk dalam cakupan penelitian ini.

5.2 Saran

Sebaiknya untuk penamaan URL lebih menggunakan kata benda yang representatif terhadap *business process* atau data yang terkait pada sebuah *endpoint*.

Daftar Pustaka

[1] K. A. Koerniawan, *Fraud Theories & Fraud Deterrence Propeller: Perkembangan Teori-Teori Fraud dan Konsep Fraud Deterrence Propeller (The DETER-E)*. Bandung: Tel-U Press, 2024.

[2] M. G. Al-Obeidallah, "The impact of design patterns on software maintainability and understandability: A metrics-based approach," *ICIC Express Letters, Part B: Applications*, vol. 12, no. 12, pp. 1111–1119, Dec. 2021, doi: 10.24507/icicelb.12.12.1111.

[3] C. Pautasso and E. Wilde, "RESTful web services," in *Proceedings of the 19th international conference on World wide web*, New York, NY, USA: ACM, Apr. 2010, pp. 1359–1360. doi: 10.1145/1772690.1772929.

[4] Sonia, A. Alsharef, P. Jain, M. Arora, S. R. Zahra, and G. Gupta, "Cache memory: An analysis on performance issues," in *Proceedings of the 2021 8th International Conference on Computing for Sustainable Global Development, INDIACom 2021*, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 184–188. doi: 10.1109/INDIACom51348.2021.00033.

[5] F. Halili and E. Ramadani, "Web Services: A Comparison of Soap and Rest Services," *Mathematical Models and Methods in Applied Sciences*, vol. 12, p. 175, 2018, [Online]. Available: https://api.semanticscholar.org/CorpusID:56080712

[6] R. Zhang, A.-M. Zhong, B. Dong, F. Tian, and R. Li, "Container-VM-PM Architecture: A Novel Architecture for Docker Container Placement," in *IEEE International Conference on Cloud Computing*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49321133

[7] A. Saha, P. Agarwal, S. Ghosh, N. Gantayat, and R. Sindhgatta, “Towards Business Process Observability,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jan. 2024, pp. 257–265. doi: 10.1145/3632410.3632435.

[8] P. Valderas and V. Torres, “Towards a Semantic Interoperability in IoTEnhanced Business Processes. An Event-Driven Solution based on Microservices,” *2023 19th International Conference on Intelligent Environments (IE)*, pp. 1–8, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:259860002

[9] G. Andreadis, C. Papaleonidas, and D. V Lyridis, “Evaluating the Operations of an LNG Shipping Company with Business Process Modelling,” 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233604269

[10] J. Cannon, “Linux Administration: The Linux Operating System and Command Line Guide for Linux Administrators,” 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:67311566

[11] B. Piedade, J. P. Dias, and F. F. Correia, “An empirical study on visual programming docker compose configurations,” *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:225067414

[12] M. Ahmed, M. M. Uddin, Md. S. Azad, and S. Haseeb, “MySQL performance analysis on a limited resource server: Fedora vs. Ubuntu Linux,” in *Spring Simulation Multiconference*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:27268304

[13] D. Kaur and M. Sachdeva, “DDoS Attacks Impact Analysis on Web Service Using Emulation,” 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:54051729

[14] M. I. Tariq, “Providing Assurance to Cloud Computing through ISO 27001 Certification: How Much Cloud is Secured After Implementing Information Security Standards,” 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:86775099

[15] T. Tanadechopon and B. Kasemsontitum, “Performance Evaluation of Programming Languages as API Services for Cloud Environments: A Comparative Study of PHP, Python, Node.js and Golang,” *2023 7th International Conference on Information Technology (InCIT)*, pp. 293–297, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:267337785

[16] N. Wei, M. N. Song, K. Xu, and C. Jiang, “A Novel WebService Architecture Based on REST,” *Adv Mat Res*, vol. 143–144, pp. 1213–1217, 2010, [Online]. Available: https://api.semanticscholar.org/CorpusID:60681503

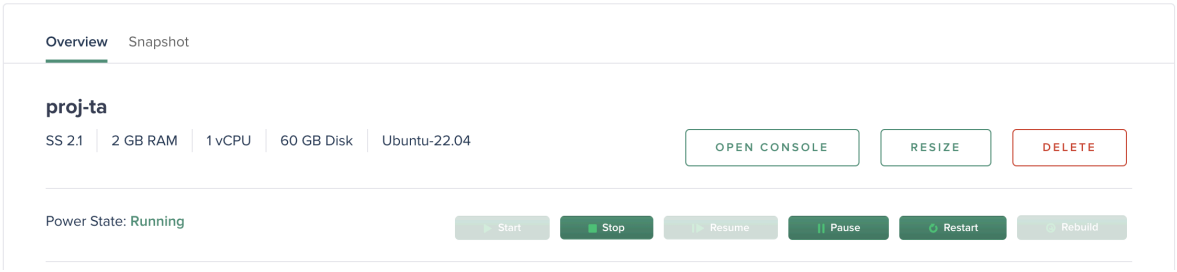
[17] M. Solanki and R. K. Sheth, “Detection of Security Vulnerabilities in Web Application using Security Headers in Python,” 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:225710498

[18] I. Dolnák and J. Litvik, “Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing,” *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 1–4, 2017, [Online]. Available: https://api.semanticscholar.org/CorpusID:23656601

[19] BizNet GioCloud, “Revolusi Adopsi Cloud Anda | Biznet Gio.” Accessed: Sep. 06, 2024. [Online]. Available: https://www.biznetgio.com/about-us

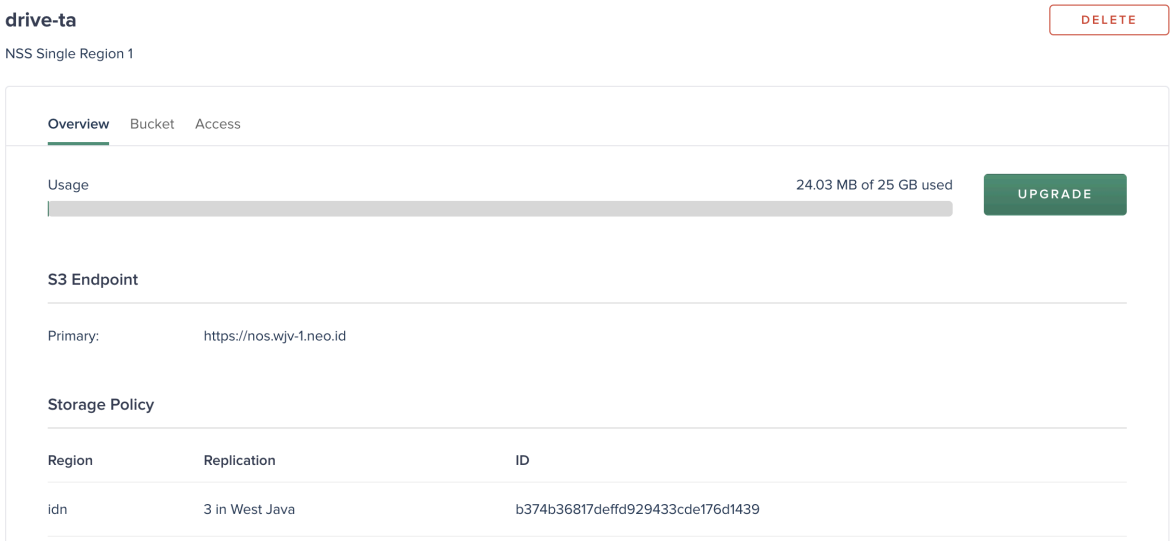
Lampiran

VPS & OS



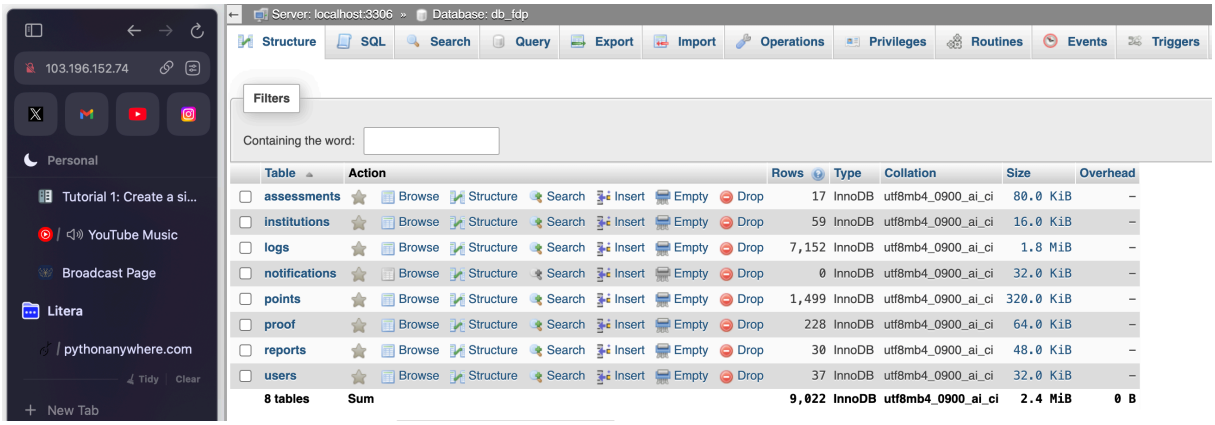
Lampiran 1 VPS

Drive



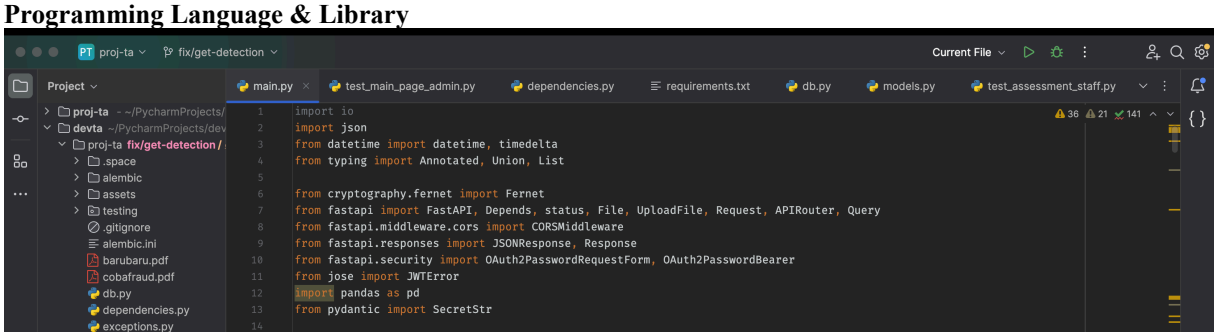
Lampiran 2 Drive

RDBMS



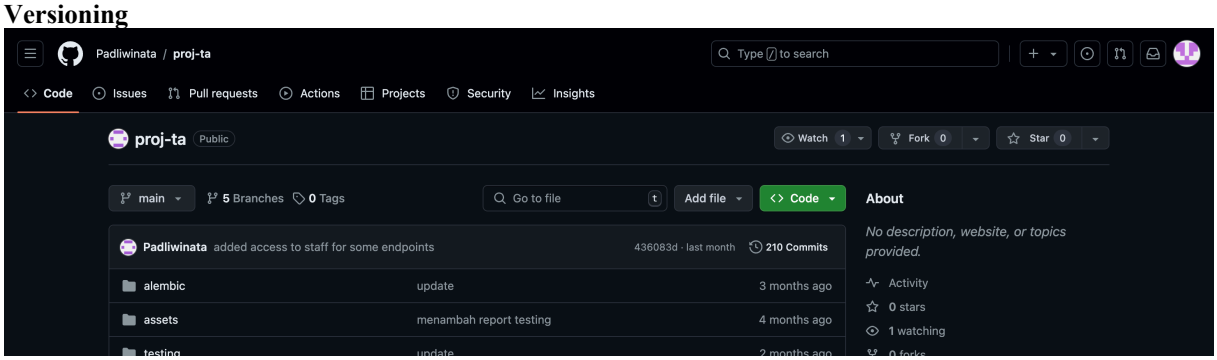
Lampiran 3 RDBMS

Programming Language & Library



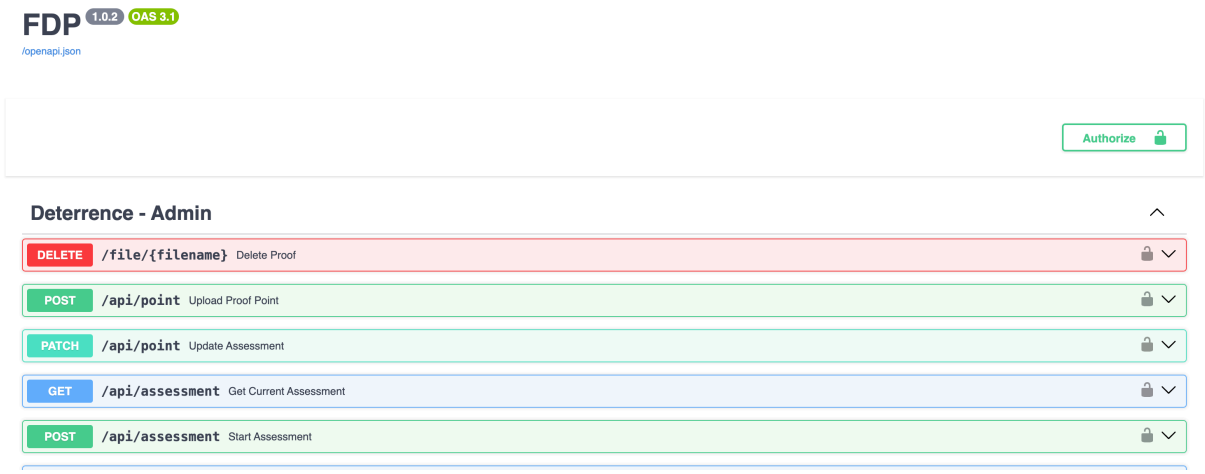
Lampiran 4 Language & Library

Versioning



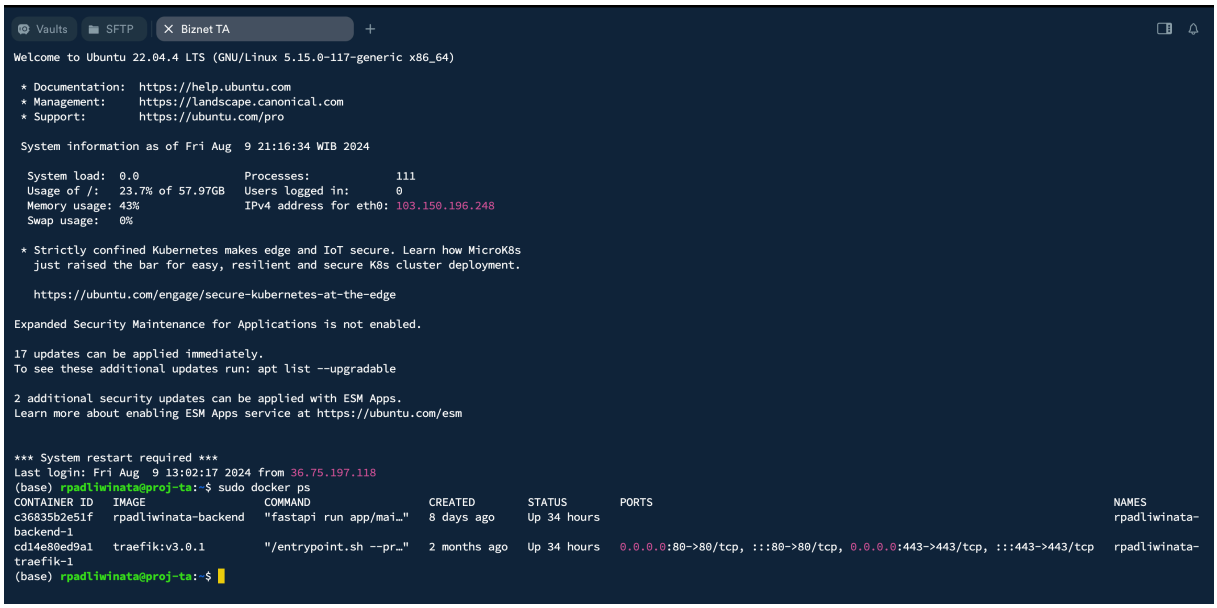
Lampiran 5 Versioning

Documentation



Lampiran 1 Swagger

Containerization



Lampiran 2 Docker