BAB II
TINJAUAN PUSTAKA

2.1 Autentikasi

Autentikasi merupakan proses konfirmasi atau validasi dari identitas individu, yang berusaha mengakses suatu sumber daya, dengan mengecek apakah identitas yang diberikan oleh individu terkait adalah benar dan dapat dibuktikan dengan membandingkan data yang diklaim oleh individu tersebut terhadap data yang ada di sistem (Boonkrong, 2021). Autentikasi adalah proses yang wajib dilakukan, untuk memverifikasi identitas dari seorang pengguna sistem, dan menghindari pengguna tidak sah untuk mengakses sumber daya sistem (Dasgupta et al., 2017).

Dalam penerapannya, terdapat 4 (empat) tipe autentikasi yang umum digunakan dalam keseharian, dimana tipe autentikasi tersebut dapat juga dikombinasikan, untuk mendapatkan manfaat dari autentikasi yang lebih kuat dan lebih baik. Tipe autentikasi yang dimaksud, antara lain:

1. Autentikasi berdasarkan apa yang pengguna sistem ketahui
2. Autentikasi berdasarkan apa yang pengguna sistem miliki
3. Autentikasi berdasarkan apa yang melekat pada diri pengguna sistem
4. Autentikasi berdasarkan posisi dimana pengguna sistem berada

Tipe nomor 1, yaitu autentikasi berdasarkan apa yang pengguna sistem ketahui, merupakan tipe autentikasi yang paling umum, dan informasi yang digunakan untuk mengautentikasi pengguna hanya boleh diketahui entitas terkait dan perlu dijaga kerahasiaannya. Beberapa yang termasuk dalam autentikasi tipe 1, antara lain kata sandi (*password*), kode keamanan, nomor identifikasi personal/*personal identification number* (PIN), dan frasa sandi (*passphrase*) (Dasgupta et al., 2017).

Tipe nomor 2, yaitu autentikasi berdasarkan apa yang pengguna sistem miliki, merupakan tipe yang mengecek kepemilikan pengguna terhadap suatu benda, dimana dalam benda tersebut terdapat fungsi atau kode unik untuk menandai individu pengguna. Beberapa alat yang umumnya digunakan dalam autentikasi tipe 2 ini, antara lain kartu pintar (*smart card*), dan token autentikasi (Boonkrong, 2021). Autentikasi tipe 2 ini pada umumnya dikenal dengan nama **autentikasi oleh kepemilikan** (Dasgupta et al., 2017).

Tipe nomor 3, yaitu autentikasi berdasarkan apa yang melekat pada diri pengguna sistem, menggunakan pendekatan fungsi humanis dan informasi fisiologis mengenai seseorang. Tipe ini menjelaskan sesuatu yang unik mengenai pengguna, dan umumnya dikenal dengan nama **autentikasi biometrik**. Tipe ini dianggap lebih canggih dibandingkan dengan dua tipe sebelumnya, dimana jika informasi autentikasi tipe 1 dan 2 dapat dibagikan, tipe 3 hampir mustahil untuk dibagikan dan dipalsukan. Karena autentikasi tipe ini melekat pada fisik seseorang, meniru dan menyalin data fisiologi seseorang untuk melewati proses autentikasi tipe ini lebih sulit atau bahkan tidak mungkin, dibandingkan dengan dua tipe sebelumnya. Dan tentu saja, data autentikasi pada tipe ini tidak dapat hilang ataupun tercuri. Adapun contoh autentikasi yang menggunakan tipe 3 ini, antara lain pengenalan wajah, sidik jari, pemindaian telapak tangan, pemindaian retina, pemindaian iris mata, pengenalan suara, dinamika ketikan, dinamika tulisan tangan, dan tanda tangan (Dasgupta et al., 2017).

Tipe nomor 4, yaitu autentikasi berdasarkan posisi dimana pengguna sistem berada, merupakan proses autentikasi yang melibatkan deteksi lokasi pengguna, untuk menentukan identitas dari pengguna yang bersangkutan. Secara umum, *global positioning system* (GPS), alamat *Internet Protocol* (IP), dan ID tower seluler digunakan untuk melakukan autentikasi tipe 4 ini. Biasanya, tipe 4 ini dikombinasikan dengan tipe autentikasi lainnya, untuk memverifikasi identitas pengguna. Pengguna akan diizinkan untuk mengakses sistem, jika pengguna memasukkan kredensial yang tepat, dan sistem mendeteksi bahwa pengguna berada di wilayah yang diizinkan. Tipe autentikasi ini juga dapat menyimpan jejak dari kapan dan dari mana pengguna masuk ke sistem. Karena autentikasi tipe 4 ini melibatkan lokasi, maka autentikasi ini dikenal dengan nama ***location based autentication*/autentikasi berbasis lokasi** (Dasgupta et al., 2017).

Dari keempat tipe autentikasi yang sudah dijelaskan sebelumnya, penelitian ini akan berfokus pada autentikasi tipe 1 dan 2, dikarenakan data yang digunakan untuk autentikasi pengguna ke sistem adalah kata sandi/*password* (autentikasi tipe 1), dan ketika pengguna sudah terautentikasi, maka *framework* Laravel akan mempergunakan *access token* (autentikasi tipe 2) sebagai tanda pengenal identifikasi pengguna terhadap layanan web eksternal.

2.2 *Uniform Resource Locator* (URL)

Uniform Resource Locator (URL) adalah nama yang terstandardisasi untuk segala sumber daya di Internet. URL menunjuk ke bagian dari informasi elektronik, yang memberitahu lokasinya, dan cara berinteraksi dengannya. URL merujuk ke lokasi sumber daya yang diperlukan peramban alias browser untuk menemukan informasi (Gourley et al., 2002). Segala macam sumber daya di Internet yang dimaksud tersebut antara lain halaman HTML, dokumen CSS, sebuah gambar, dan lain-lain (MDN Web Docs, 2021b).

2.2.1 Anatomi URL

URL memungkinkan untuk mengakses sumber daya di Internet, dari berbagai macam skema, protokol, dan alamat yang berbeda, menggunakan satu bentuk yang seragam dan konsisten. Dikutip dari **MDN Web Docs** (2021b), bentuk atau anatomi dari sebuah URL yang umum digunakan dalam mengakses situs web, dapat dideskripsikan melalui Gambar 2.1 berikut.

Gambar 2.1 Anatomi URL

Secara sederhana, *scheme* atau skema adalah pengenal utama yang memberitahu cara untuk mengakses sumber daya tertentu, atau dengan kata lain, memberitahu aplikasi mengenai protokol apa yang akan berkomunikasi dengannya (Gourley et al., 2002). Untuk situs web, protokol yang digunakan biasanya adalah HTTP, atau HTTPS untuk versi yang aman (MDN Web Docs, 2021b).

Kemudian ada *authority*/otoritas, yang berisi nama domain (*domain name*) dan *port*, yang dipisahkan oleh titik dua (:). Domain menunjuk ke server web yang sedang diminta; biasanya ini adalah nama domain, tapi alamat IP juga dapat digunakan, walaupun agak kurang diminati; beberapa literatur menyebut nama domain dengan kata *host*, yang dimana hal ini sebenarnya merujuk ke hal yang sama, yaitu mesin di Internet yang memiliki sumber daya yang dimaksud (Gourley et al., 2002). Sedangkan *port* sendiri adalah "jembatan" teknis untuk mengakses sumber daya di server web. *Port* ini biasanya tidak perlu dimasukkan, jika server web menggunakan nomor *port* standar (80 untuk HTTP, dan 443 untuk HTTPS) dalam

memberikan akses ke sumber dayanya (MDN Web Docs, 2021b).

Jalur/*path*, yang menuju ke berkas/sumber daya, sesuai dengan namanya, adalah jalur yang menunjuk ke sumber daya yang dituju di server web. Di awal mula hadirnya situs web, jalur/*path* seperti `/path/to/my/file.html`, memang menunjukkan lokasi berkas fisik yang berada di sistem berkas server web. Jaman sekarang, jalur/*path* ini kebanyakan berupa abstraksi yang ditangani oleh server web, tanpa kehadiran fisik yang nyata (MDN Web Docs, 2021b).

Bagian unik lainnya dari penyusun URL, yaitu parameter, adalah komponen ekstra yang dikirimkan ke server web. Parameter adalah komponen bertipe pasangan kunci/nilai (*key/value pairs*) yang diformat menjadi `kunci=nilai`, dan antar parameter dipisahkan oleh simbol "&". Server web dapat menggunakan parameter tersebut untuk melakukan hal lain yang dibutuhkan, sebelum mengembalikan sumber daya yang diminta (MDN Web Docs, 2021b). Beberapa literatur menyebut bagian ini dengan *query string*, yang seharusnya mereferensikan hal yang sama (Gourley et al., 2002).

Bagian terakhir dari anatomi URL diatas adalah *anchor*, atau *fragment*. *Anchor*/*fragment*, yang diawali dengan tanda "#" ini adalah penanda yang merujuk ke satu bagian (*section*) dari keseluruhan sumber daya yang diminta, dengan memberikan petunjuk detail dan arahan untuk menuju ke satu bagian yang ditandai olehnya (MDN Web Docs, 2021b). Misalkan ada satu berkas teks yang panjang di sebuah server web, dengan penanda bagian (*section*) didalamnya. Permintaan URL tanpa menyertakan *anchor* ke server web akan mengembalikan keseluruhan berkas teks yang panjang tersebut, dan titik mulanya akan berada pada awal dokumen yang dimaksud, tetapi jika *anchor* ditambahkan pada URL, maka peramban akan merujuk ke satu bagian kecil dari keseluruhan sumber daya terkait, dalam kasus ini, berkas teks yang panjang tersebut (Gourley et al., 2002). Perlu diingat bahwa bagian yang terletak setelah tanda "#" pada URL tidak pernah dikirimkan ke server dalam permintaan yang dibuat (MDN Web Docs, 2021b).

Hampir tidak ada URL yang memuat keseluruhan komponen penyusun anatominya sekaligus. Tiga bagian paling penting dari sebuah URL adalah skema, *host*/nama domain, dan jalur/*path* (Gourley et al., 2002).

Dalam penelitian ini, skema yang akan digunakan dalam URL adalah skema

HTTP versi aman, alias HTTPS, yang sudah menjadi hal umum dalam beberapa tahun terakhir, dibuktikan dengan meningkatnya jumlah situs web yang berjalan di HTTPS (W3Techs, 2021).

2.3 *Hypertext Transfer Protocol* (HTTP)

Hypertext Transfer Protocol (HTTP) adalah protokol level aplikasi untuk sistem hipermedia informatif yang terdistribusi dan kolaboratif. Protokol ini merupakan protokol generik dan *stateless*; dapat dipergunakan untuk kegunaan lainnya selain untuk hiperteks, seperti untuk server nama (*name server*) dan sistem manajemen distribusi objek, melalui perpanjangan terhadap metode permintaannya, kode galatnya, dan kepala pesannya (Fielding et al., 1999).

HTTP merupakan protokol yang digunakan oleh program untuk berkomunikasi melalui *World Wide Web* (WWW). Peramban/*browser web*, server web, dan aplikasi web terkait lainnya, semua berbicara ke dan dari satu sama lain melalui HTTP, yang sudah menjadi bahasa umum untuk Internet global modern (Gourley et al., 2002). Dari kemunculan awalnya yang hanya sebagai kata kunci tunggal dan penanda jalur dokumen, HTTP telah menjadi protokol pilihan yang tidak hanya berlaku untuk peramban web, tetapi juga untuk hampir semua perangkat lunak dan aplikasi perangkat keras yang terhubung ke Internet (Grigorik, 2017).

Ada beberapa versi dari HTTP itu sendiri, antara lain versi 0.9 yang menjadi perintis di tahun 1991, diikuti versi 1.0 yang menjadi standar di tahun 1996. Versi 1.1 menyusul untuk menyempurnakan versi 1.0 di tahun 1997 (Grigorik, 2017). Internet mengalami perkembangan luar biasa dalam kurun waktu 18 tahun sejak 1997, hingga akhirnya *HTTP Working Group* (HTTP-WG) yang merupakan bagian dari *Internet Engineering Task Force* (IETF) meratifikasi versi terbaru dari protokol HTTP, yaitu versi 2.0, melalui RFC 7540 pada tahun 2015 (Belshe et al., 2015).

Tahun 2020, ada 49,22% halaman rumah (*homepage*) versi *desktop* dan 50,05% halaman rumah versi perangkat bergerak (*mobile*) yang menggunakan protokol HTTP versi 1.1. Di saat yang sama, terdapat 49,97% halaman rumah versi

desktop, dan 49,28% halaman rumah versi *mobile* yang berjalan di protokol HTTP versi 2.0 (Galloni et al., 2020).

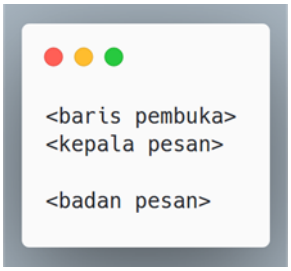
Agar penelitian ini lebih terstruktur, versi HTTP yang akan digunakan dalam penelitian ini adalah HTTP versi 1.1.

2.3.1 Pesan HTTP (*HTTP message*)

Agar dapat berkomunikasi satu dengan yang lainnya, aplikasi yang ingin dapat saling terhubung, harus memiliki standar baku komunikasi yang berlaku dalam protokolnya. Hal ini juga berlaku dalam protokol HTTP; aplikasi yang ingin berbicara dengan aplikasi lainnya dalam protokol HTTP, memerlukan format standar pesan yang dapat dipahami kedua pihak yang saling berkomunikasi. Dalam hal ini, sebuah pesan yang bernama pesan HTTP/*HTTP message*. Pesan HTTP ini berupa kumpulan blok data yang dikirim di antara aplikasi HTTP yang saling berkomunikasi. Blok data tersebut dimulai dengan beberapa teks informasi meta, yang menjelaskan isi badan pesan HTTP tersebut dan artiannya, disusul dengan data tambahan yang opsional (Gourley et al., 2002).

Tiap pesan permintaan ataupun respons HTTP ini terdiri dari tiga bagian utama: sebuah baris pembuka yang menjelaskan pesannya, sebuah blok kepala (*header*) yang memuat atribut pesan, dan blok badan (*body*) yang bersifat opsional dan memuat data. Baris pembuka dan kepala berupa teks berformat ASCII, yang dipisahkan oleh baris. Tiap akhir baris diakhiri oleh dua karakter susunan penutup baris (*end-of-line sequence*), tersusun atas *carriage return* (ASCII 13) dan karakter *line-feed* (ASCII 10); kedua karakter ini biasanya tertulis sebagai CRLF. Perlu diingat juga walaupun spesifikasi resmi HTTP untuk mengakhiri baris adalah dengan menggunakan CRLF, aplikasi yang canggih seharusnya juga menerima karakter *line-feed* (LF) untuk menutup baris. Beberapa aplikasi HTTP tua atau separuh rusak tidak selalu mengirimkan karakter *carriage return* (CR) dan *line-feed* (LF) bersamaan (Gourley et al., 2002). Kepala dan badan pesan HTTP dipisahkan oleh satu baris kosong (*empty line*) untuk menandakan batas akhir dari kepala pesan HTTP (Fielding et al., 1999).

Format umum untuk pesan HTTP ditampilkan dalam Gambar 2.2 berikut.

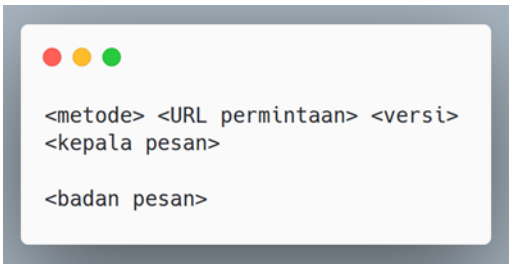


Gambar 2.2 Format umum pesan HTTP

2.3.2 Permintaan dan Respons HTTP

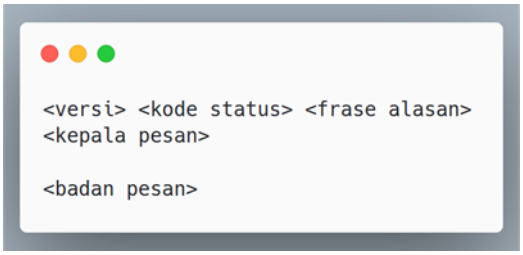
Semua pesan HTTP terbagi dalam dua jenis: pesan permintaan dan pesan respons. Pesan permintaan meminta aksi dari sebuah server web, dan pesan respons memberikan jawaban dari server web, mengenai hasil yang diminta dari pesan permintaan sebelumnya. Baik pesan permintaan maupun respons memiliki struktur dasar pesan HTTP yang sama (Gourley et al., 2002).

Walaupun memiliki struktur dasar pesan yang sama, isi dalam pesan permintaan dan pesan respons tentu saja berbeda, dimana format lengkap untuk pesan permintaan dapat dirumuskan sebagai berikut dalam Gambar 2.3.



Gambar 2.3 Format pesan permintaan HTTP

Sedangkan untuk format lengkap dari pesan respons HTTP dijelaskan dalam Gambar 2.4 berikut.



Gambar 2.4 Format pesan respons HTTP

Terlihat bahwa perbedaan utama terletak di bagian baris pembuka pesan; pesan permintaan mengirimkan metode, URL permintaan, dan versi dari HTTP yang digunakan pada baris pembukanya; sedangkan pesan respons mengirimkan

versi HTTP, kode status, dan frase alasan (*reason phrase*) yang berupa penjelasan dari kode status yang ikut dikirimkan (Gourley et al., 2002).

2.3.3 Metode HTTP

Untuk mendapatkan hasil yang sesuai dari permintaan yang dikirimkan ke server web melalui protokol HTTP, maka permintaan HTTP harus menggunakan metode HTTP yang sesuai. Metode HTTP adalah sekumpulan kata kerja (*verb*) untuk memberitahu protokol HTTP tentang sesuatu yang harus dikerjakan oleh server web dalam merespons permintaan HTTP. Terdapat 39 metode HTTP yang terdaftar secara resmi oleh *Internet Assigned Numbers Authority* (IANA) dalam registrinya (IANA, 2017); tetapi hanya 6 (enam) metode HTTP yang akan digunakan untuk mengakomodir gaya arsitektur REST dalam penelitian ini, antara lain :

1. **GET**

   Metode GET adalah permintaan paling umum yang dikirim ke server. Metode ini umumnya adalah untuk meminta server untuk mengirimkan sebuah sumber daya (Gourley et al., 2002). Metode GET adalah mekanisme primer untuk mengambil informasi; oleh karena itu, ketika banyak orang mengatakan untuk mengambil beberapa informasi yang bisa diidentifikasi melalui protokol HTTP, yang mereka maksudkan adalah mengambil informasi tersebut dengan metode GET (Fielding & Reschke, 2014).

2. **HEAD**

   Metode HEAD memiliki cara kerja yang mirip dengan metode GET, tetapi bedanya adalah metode HEAD hanya mengembalikan kepala (*header*) pesan HTTP tanpa disertai badan (*body*) pesan (Gourley et al., 2002). Server harus mengirimkan kolom kepala/*header* yang sama untuk permintaan GET ataupun HEAD ke alamat terkait. Biasanya metode ini digunakan untuk mengecek metadata dari suatu sumber daya, tanpa mengirimkan isi lengkap dari sumber daya terkait; umumnya sebagai upaya untuk menguji tautan hiperteks dalam rangka

mengecek validitas, aksesibilitas, dan perubahan terbaru (Fielding & Reschke, 2014).

3. **POST**

   Metode POST digunakan untuk meminta server yang dituju, agar menerima berkas yang dikirimkan dalam badan pesan HTTP, untuk dijadikan sebuah entri baru di sistem, yang dapat dikenali dan diakses dari URL yang merujuk kepadanya (Fielding et al., 1999). Metode ini umumnya digunakan untuk beberapa hal, seperti mengirimkan data dari formulir HTML agar diproses oleh prosesor data; mengirimkan pesan ke papan buletin (*bulletin board*), *mailing list*, blog; membuat sumber daya baru yang nantinya akan dikenali oleh server, dan menambahkan data ke implementasi sumber daya yang telah ada (Fielding & Reschke, 2014).

4. **PUT**

   Metode PUT digunakan untuk mengirimkan berkas dokumen ke server, kebalikan dari metode GET. Maksud semantik dari metode PUT ini adalah meminta server untuk mengambil data dari badan pesan permintaan HTTP, dan menyimpannya sebagai dokumen baru, atau jika dokumen yang dimaksud sudah ada, maka isi badan pesan permintaan tersebut akan menimpa data yang sudah ada di sistem tersebut (Gourley et al., 2002).

5. **PATCH**

   Metode PATCH digunakan untuk meminta server membuat perubahan, sesuai pada serangkaian perubahan yang didefinisikan dalam sebuah dokumen media yang ikut serta dalam pesan permintaan HTTP tersebut, pada sumber daya yang berada padanya dan dapat diidentifikasi oleh URL. Jika URL yang dituju tidak menunjuk pada sebuah sumber daya, maka server mungkin dapat membuat sumber daya baru, tergantung dari dokumen tambalan (*patch document*) yang disertakan, permisi atau izin dari pengguna, dan sebagainya (Dusseault & Snell, 2010).

Perbedaan metode ini dengan metode PUT terletak pada bagaimana server memproses entitas untuk mengubah sumber daya terkait: entitas dokumen yang dibawa oleh pesan HTTP dengan metode PUT, akan dianggap sebagai versi modifikasi dari entitas sumber daya yang sudah ada di server, sedangkan pada metode PATCH, entitas yang dibawa oleh pesan HTTP terkait adalah serangkaian instruksi, yang menjelaskan cara untuk mengubah sebuah entitas yang berada di server. Metode PATCH mempengaruhi sumber daya yang dikenali dengan URL, dan mungkin saja memiliki efek samping terhadap sumber daya lainnya, yang tidak ditunjuk langsung oleh sebuah URL (Dusseault & Snell, 2010).

6. **DELETE**

   Metode DELETE bekerja seperti namanya: meminta server untuk menghapus sumber daya di server, yang dikenali dari URL permintaan. Tetapi, aplikasi klien tidak diberikan kepastian apakah perintah penghapusan benar-benar dilakukan oleh server. Hal ini terjadi, karena spesifikasi resmi dari HTTP mengizinkan server untuk melakukan *overriding* terhadap permintaan, tanpa memberitahu klien (Gourley et al., 2002).

Tetapi dalam sebagian besar kasus, terutama dalam arsitektur internal *framework* Laravel, metode PUT dan PATCH dianggap memiliki cara kerja yang persis sama, yaitu dengan menganggap entitas dokumen yang ikut serta dalam permintaan HTTP terkait, adalah versi modifikasi dari entitas sumber daya yang sudah ada di server sebelumnya.

2.3.3.1 Properti Metode HTTP

Metode HTTP yang telah disebutkan diatas, dan juga metode HTTP lainnya, semuanya terikat dalam properti umum yang merangkul beberapa metode terkait dalam satu grup yang memiliki kesamaan. Properti yang dimaksud tersebut, antara lain :

1. Metode aman (*safe method*)

Sebuah metode dapat dikatakan aman, jika definisi semantiknya, secara umum, adalah metode permintaan hanya-baca (*read only*); dengan kata lain, metode yang tidak melakukan perubahan pada keadaan/*state* dari server yang dituju (Fielding & Reschke, 2014).

2. Metode idempoten (*idempotent method*)

   Sebuah metode dapat dikatakan sebagai metode idempoten, jika permintaan yang dibuat dengan metode tersebut selalu mengembalikan hasil yang sama, jika permintaan tersebut dilakukan sekali, ataupun berkali-kali. Walaupun demikian, properti idempoten ini hanya berlaku untuk segala sesuatu yang diminta dalam permintaan HTTP terkait; server bebas untuk mencatat log tiap permintaan yang masuk, menyimpan sejarah revisi dan sebagainya, selama tiap hal yang dimaksud tidak mengganggu status idempoten tersebut (Fielding & Reschke, 2014).

3. *Cacheable method*

   Sebuah metode dapat didefinisikan sebagai *cacheable*, untuk menunjukkan fakta bahwa respons dari permintaan terkait, dapat disimpan untuk penggunaan di masa mendatang (Fielding & Reschke, 2014).

2.3.4 Kode Status HTTP

Setiap respons atau balasan dari permintaan HTTP, selalu disertai dengan kode status. Kode status ini merepresentasikan cara yang mudah bagi klien untuk memahami hasil transaksi yang diminta sebelumnya, dan termuat dalam 5 (lima) kelas kategori utama (Gourley et al., 2002).

Kode status ini berupa tiga digit bilangan bulat, yang tiap digitnya memiliki arti tertentu. Digit pertama dari kode status itu menunjukkan kelas kategorinya, dan dua sisanya adalah penjelasan rinci dari kelas yang dimaksud. Kode status HTTP ini sendiri dapat diperpanjang (*extensible*) dari kelas yang ada.

Setiap klien HTTP tidak wajib memahami seluruh kode status HTTP yang teregister, tetapi setiap klien HTTP wajib memahami kelas kode status HTTP, dan menangani kode yang tidak dikenal menjadi sama dengan x00 dari kelas terkait. Semua respons yang menggunakan kode status HTTP yang tidak dikenal, tidak

boleh di-*cache* oleh klien (Fielding & Reschke, 2014).

Menurut dokumen RFC 7231 yang ditulis oleh **Fielding & Reschke** (2014), terdapat lima kelas kategori yang terdaftar untuk kode status HTTP, antara lain:

1. **Informasional (*informational*)**

   Kode status yang berada di kelas ini, memiliki digit pertama berupa angka 1 (satu). Rentang angka yang valid di kelas ini bernilai 100-199. Sesuai namanya, kelas ini berisikan kode yang menyampaikan informasi umum ke klien, misalnya kode **100 Continue**, **101 Switching Protocol**.

2. **Sukses (*successful*)**

   Kode status yang berada di kelas ini, memiliki digit pertama berupa angka 2 (dua). Rentang angka yang valid di kelas ini bernilai 200-299. Kode yang berada di kelas ini digunakan untuk menyampaikan pesan sukses terhadap permintaan klien, sesuai dengan jenis permintaan yang dilakukan klien sebelumnya; dan yang umum digunakan dari kelas ini adalah **200 OK**, **201 Created**, dan **204 No Content**.

3. **Pengalihan (*redirection*)**

   Kode status yang berada di kelas ini, memiliki digit pertama berupa angka 3 (tiga). Rentang angka yang valid di kelas ini bernilai 300-399. Kelas kode ini memberitahu klien untuk beralih ke lokasi alternatif dari sumber daya yang diminta, atau memberikan respons alternatif alih-alih sumber daya yang diminta. Jika sumber daya yang diminta sudah pindah lokasi, sebuah kode status dari kelas pengalihan dan kepala/*header* bernama "Location" pada pesan respons HTTP dapat dikirim ke klien untuk memberitahu mengenai perpindahan sumber daya tersebut, beserta lokasi barunya (Gourley et al., 2002). Kode yang umum digunakan adalah **301 Moved Permanently**, **302 Found**, dan **304 Not Modified**.

4. **Galat klien (*client error*)**

   Kode status yang berada di kelas ini, memiliki digit pertama berupa angka 4 (empat). Rentang angka yang valid di kelas ini bernilai 400-499. Kelas kode ini berfungsi sesuai namanya: memberitahukan kesalahan atau galat yang terjadi di sisi klien, misalnya pesan permintaan yang

tidak sesuai format yang dikenal server (Gourley et al., 2002). Kode yang umum digunakan dari kelas ini adalah **400 Bad Request**, **401 Unauthorized**, **403 Forbidden**, dan **404 Not Found**.

5. **Galat server (*server error*)**

   Kode status yang berada di kelas ini, memiliki digit pertama berupa angka 5 (lima). Rentang angka yang valid di kelas ini bernilai 500-599. Kebalikan dari kelas sebelumnya, kode pada kelas ini menyampaikan kesalahan atau galat yang terjadi dari sisi server; misalnya melampaui batas kemampuan pemrosesan server (Gourley et al., 2002). Kode pada kelas ini yang umum ditemui penggunaannya, antara lain **500 Internal Server Error**, **502 Bad Gateway**, dan **503 Service Unavailable**.

2.3.5 Kepala (*header*) Pesan HTTP

Kepala (*header*) dan metode bekerja sama untuk menentukan apa yang harus dikerjakan oleh server dan klien. Ada kepala yang memiliki fungsi spesifik, dan ada kepala yang memiliki fungsi umum (Gourley et al., 2002). Secara garis besar, kepala pesan HTTP ini terbagi dalam lima kelas utama, antara lain:

1. *Header* umum

   Header ini menyediakan informasi sederhana tentang sebuah pesan HTTP, seperti informasi jenis koneksi, *timestamp*, *caching*, dan sebagainya (Gourley et al., 2002). Contoh dari *header* di kelas ini antara lain **Connection**, **MIME-Version**, **Trailer**, **Date**, **Age**, **Transfer-Encoding**, dan **Cache-Control**.

2. *Header* permintaan

   Header ini hanya berlaku di pesan permintaan HTTP. Biasanya, *header* jenis ini memberitahu tentang siapa pengirim pesan terkait, apa yang dikirim, dimana asal pengirimnya, dan apa saja kemampuan yang dimiliki klien untuk mengolah respons dari server (Gourley et al., 2002). Contoh dari *header* ini adalah **User-Agent**, **Host**, dan **Referrer**.

3. *Header* respons

   Header pada kelas ini berguna untuk memberitahukan klien informasi tambahan mengenai sumber daya yang dikembalikan oleh server,

seperti siapa pengirim respons dan kapabilitas dari pengirim (Gourley et al., 2002). Contoh dari *header* di kelas ini adalah **Date**, **Age**, **Retry-After**, **Server**, **Vary**, **Proxy-Authenticate**, **Set-Cookie**, dan **WWW-Authenticate**.

4. *Header* entitas

   Header di kelas ini menjelaskan mengenai isi (*payload*) yang dibawa dalam pesan respons HTTP. Karena pesan permintaan dan respons dapat membawa entitas, maka *header* ini dapat hadir diantara kedua jenis pesan tersebut (Gourley et al., 2002). Contoh dari *header* di kelas ini, antara lain **Allow**, **ETag**, **Expires**, **Last-Modified**, **Location**, **Content-Security-Policy**, **Content-Encoding**, **Content-Length**, **Content-Language**, dan **Content-Type**.

5. *Header* perpanjangan/ekstensi

   Header yang berada di kelas ini biasanya berupa *header* nonstandar yang dibuat oleh pengembang aplikasi tertentu, tetapi belum ditambahkan ke spesifikasi HTTP resmi. Program HTTP perlu bertoleransi dan meneruskan *header* ekstensi tersebut ke rangkaian aplikasi yang ada setelahnya, sekalipun mereka tidak paham arti dari *header* tersebut (Gourley et al., 2002).

2.3.5.1 *Header* Keamanan

HTTP memiliki dukungan bawaan untuk mekanismie autentikasi pengguna dengan strategi *challenge and response*. Cara ini dinilai dapat sedikit membantu keamanan transaksi data dengan mewajibkan pengguna untuk mengirim kredensialnya, sebelum mengakses properti atau sumber daya terlindungi. Dalam praktik menjaga keamanan dan sebagai media autentikasi permintaan HTTP, terdapat 2 (dua) *header* yang umum digunakan untuk hal ini, yaitu *header* Authorization dan Cookie (Gourley et al., 2002). Walaupun keduanya umum dipergunakan untuk melakukan autentikasi permintaan HTTP, hanya *header* Authorization yang berfungsi untuk mengirimkan kredensial pengguna, sedangkan *header* Cookie hanya dipergunakan untuk mengirimkan pengenal sesi (*session*

identifier) pengguna ke server, dimana *state* kondisi autentikasi dan kredensial pengguna disimpan oleh server (Gourley et al., 2002).

Untuk penelitian ini, penulis berfokus pada *header* Authorization saja, dikarenakan mekanisme autentikasi permintaan pada server API Papon.id mensyaratkan penggunaan *header* tersebut. Selain itu, penggunaan header Cookie sudah diatur secara otomatis oleh *framework* Laravel dan peramban yang terkoneksi dengannya.

2.4 Autentikasi Permintaan HTTP

Menurut rancangannya, permintaan HTTP adalah permintaan *stateless*, yang berarti server tidak menyimpan *state* dari tiap permintaan yang terjadi, sehingga server tidak mengetahui apakah permintaan sebelum dan setelahnya merupakan permintaan terautentikasi, serta berasal dari klien yang sama atau tidak. Dua permintaan yang berasal dari klien yang sama dan menuju server yang sama, dapat dianggap dua permintaan berbeda. Implikasi dari keadaan ini adalah, server tidak dapat mengenali pengguna secara langsung dari tiap permintaan yang masuk, tanpa melibatkan mekanisme tambahan yang memungkinkan hal itu untuk terjadi.

Oleh karena itu, terdapat dua teknik yang umum digunakan oleh server untuk mengidentifikasi apakah suatu permintaan itu dibuat oleh pengguna yang terdaftar di sistem atau tidak. Dua teknik itu antara lain:

1. Autentikasi berbasis sesi (*session based authentication*)

   Autentikasi berbasis sesi bekerja dengan cara menyimpan data pengguna yang terautentikasi, ke memori atau penyimpanan server. Pada autentikasi berbasis sesi, server akan membuat sesi baru ketika pengguna berhasil masuk (*login*) ke sistem. Sebuah *cookie*, yang berisi tanda pengenal sesi, kemudian dikirimkan dari server ke klien, sebagai balasan dari permintaan *login* sebelumnya. *Cookie* ini akan dikirimkan kembali ke server dalam setiap permintaan HTTP dari klien tersebut, dan server akan membandingkan pengenal sesi yang ada didalam *cookie* tersebut dengan sesi berisi data pengguna terautentikasi yang disimpan di server, dan server akan merespon dengan kondisi *state* yang sesuai dari pengguna terkait (Hsu, 2018). Dalam beberapa

literatur, autentikasi berbasis sesi ini juga disebut sebagai autentikasi berbasis kuki (*cookie based authentication*) (Dulanga, 2021).

2. Autentikasi berbasis token (*token based authentication*)

   Autentikasi berbasis token bekerja hampir mirip seperti autentikasi berbasis sesi; bedanya adalah, data pengguna yang sudah berhasil *login* ke sistem tidak disimpan ke memori atau medium penyimpanan data di server, tetapi disimpan ke sebuah token, yang mengikuti format ***JSON Web Token*** (JWT). Token JWT ini kemudian ditandatangani oleh server dengan sebuah kunci rahasia (*secret*) dan dikirimkan ke klien untuk disimpan di sisi klien. Klien akan menyimpan token ini di penyimpanan lokal (*local storage*) pada aplikasi klien tersebut, dan akan dikirimkan kembali ke server dalam setiap kepala (*header*) permintaan HTTP dari klien terkait melalui *header* **Authorization**. Server yang menerima permintaan HTTP yang memuat token tersebut akan mengecek apakah token tersebut valid, dengan membandingkan *signature hash* dari token tersebut dengan kunci rahasia (*secret*) milik server (Hsu, 2018).

Perbedaan utama dari kedua strategi autentikasi tersebut adalah, pada autentikasi berbasis sesi, *state* dari pengguna terautentikasi akan disimpan pada sisi server (*server side*), dan pada autentikasi berbasis token, *state* pengguna tersebut akan disimpan di sisi klien (*client side*) (Hsu, 2018).

Pada praktiknya, Papon.id sendiri menggunakan kedua metode ini, dimana autentikasi berbasis sesi digunakan oleh *framework* Laravel yang bertindak sebagai aplikasi web menghadap-pengguna (*user-facing*), untuk menangani pengguna terautentikasi, sedangkan autentikasi berbasis token digunakan oleh layanan REST API Papon.id untuk mengautentikasi permintaan dari klien, yang berupa perangkat *mobile*, dan website yang dibangun diatas *framework* Laravel.

2.5 PHP

PHP adalah singkatan rekursif dari *PHP: Hypertext Preprocessor*, merupakan sebuah bahasa pemrograman *general-purpose* dengan kode sumber

terbuka (*open source*), yang umumnya digunakan dalam pengembangan web dan dapat disematkan dalam kode HTML (The PHP Working Group, 2021b).

Bahasa PHP diciptakan oleh Rasmus Lerdorf, seorang *programmer* dari Denmark pada tahun 1994, dan mulai dipublikasikan untuk umum pada sebuah forum di internet pada tahun 1995 dengan nama *Personal Home Page (PHP) Tools* (Lerdorf, 1995).

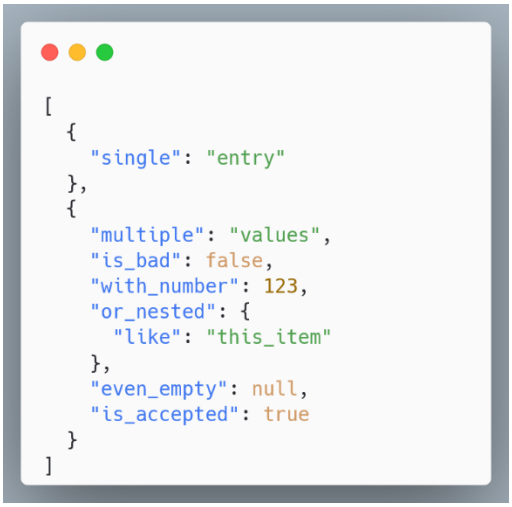
Bahasa PHP sendiri pada umumnya digunakan sebagai bahasa sisi server (*server-side*) untuk pemrosesan halaman web, seperti mengumpulkan data formulir, mengirim dan menerima *cookie*, serta menciptakan tampilan halaman web dinamis. Selain dari kegunaannya dalam sisi server, PHP sendiri dapat digunakan dalam aplikasi *command line*. Hal ini memungkinkan kode sumber PHP untuk dijalankan pada sistem operasi secara langsung menggunakan *interpreter* PHP yang sudah diinstal, tanpa memerlukan server web ataupun peramban web. Bahasa PHP juga dapat digunakan untuk mengembangkan aplikasi *desktop*, walaupun mungkin PHP bukan bahasa terbaik yang cocok dalam bidang ini (The PHP Working Group, 2021a).

Penggunaan bahasa pemrograman PHP pada penelitian ini menjadi hal penting, karena kode sumber *framework* Laravel ditulis menggunakan bahasa ini. Dalam penelitian ini, penulis akan menggunakan bahasa pemrograman PHP versi 8.0 sebagai standar acuan teknis untuk mengembangkan pustaka *user provider* ini.

2.6 *JavaScript Object Notation* (JSON)

JavaScript Object Notation (JSON) merupakan sebuah format data yang independen, dan menggambarkan objek JSON sebagai daftar properti yang bisa dibaca oleh manusia. Walaupun dibuat dari bagian kecil yang masih berkaitan dengan JavaScript, yaitu rancangan desainnya, JSON dapat di-*parse* menjadi objek data dengan mudah oleh banyak bahasa pemrograman (Friesen, 2019).

Format data yang digunakan oleh JSON merupakan format data yang umum digunakan, terutama didalam bahasa pemrograman dalam keluarga bahasa C, seperti C/C++, C#, Java, dan tentu saja JavaScript. Melihat kenyataan ini, JSON ini ideal digunakan sebagai format bahasa pertukaran data (JSON.org, 2022).



Gambar 2.5 Contoh format data JSON

Karena server dan klien aplikasi Papon.id terpisah menurut standar gaya arsitektur REST, perlu ada satu format pertukaran data, yang memungkinkan mereka dapat berkomunikasi dengan baik. Akibatnya, JSON dipilih menjadi format pertukaran data yang cocok, karena sifatnya yang ringan, universal, dan portabel.

2.7 Composer

Composer adalah sebuah aplikasi perangkat lunak yang berfungsi sebagai manajer paket untuk proyek berbasis PHP. Aplikasi ini berguna untuk membantu pengembang yang bekerja dengan bahasa pemrograman PHP, dengan cara menginstal paket pustaka, yang ingin dipakai dalam proyek perangkat lunak yang sedang dikembangkan (Adermann & Boggiano, 2022). Dengan menggunakan Composer, selain untuk memudahkan pekerjaan, diharapkan juga dapat membuat struktur proyek PHP menjadi lebih rapi dan mudah dirawat (Shinta, 2022).

Menurut Shinta (2022), untuk mengunduh paket pustaka yang dibutuhkan oleh pengembang, Composer melakukan beberapa langkah kerja sebagai berikut:

1. Composer mencari paket pustaka yang dimaksud oleh pengembang di repositori Packagist, yaitu sebuah lokasi sentral penyimpanan paket pustaka Composer.
2. Jika paket pustaka yang dimaksud ditemukan, maka Composer akan memeriksa beberapa hal sebelum mengunduh, seperti:
    a. Apakah pustaka yang diminta membutuhkan pustaka lainnya sebagai dependensi?

b. Apakah versi PHP yang diinstal pada perangkat, memenuhi versi yang disyaratkan oleh pustaka tersebut?
c. Apakah ada modul PHP yang dibutuhkan oleh pustaka tersebut? Apakah sudah terinstal pada perangkat?

3. Jika proses pengecekan telah selesai, dan tidak ada halangan ditemukan, maka Composer akan mengunduh paket pustaka tersebut, dan menyimpannya pada folder "vendor" yang terletak dalam folder kerja aktif (*current working directory*) dimana Composer dipanggil.

Penulis akan mempergunakan Composer sebagai manajer paket dependensi untuk mengembangkan modul pustaka *user provider*, karena *framework* Laravel membutuhkan Composer untuk mengunduh pustaka *framework*-nya, baik pustaka inti maupun pustaka pihak ketiga yang dipergunakan oleh Laravel.

2.8 Git

Git merupakan perangkat lunak sistem kontrol versi/*version control system* (VCS) yang dirilis pertama kali oleh Linus Torvalds pada tahun 2005. Perangkat lunak ini bekerja dengan cara membuat catatan versi tentang perubahan yang terjadi didalam repositori proyek. Tidak seperti pada sistem kontrol versi lainnya, Git mengadopsi gaya sistem terdistribusi, alih-alih mempergunakan strategi terpusat seperti sistem kontrol versi lainnya. Hal ini memungkinkan setiap pengguna Git memiliki repositori sendiri yang dapat terpisah dari induknya (Liberty, 2021).

2.8.1 GitHub

GitHub merupakan salah satu penyedia layanan *hosting* untuk Git, dimana layanan ini memungkinkan pengguna Git untuk menyimpan repositori mereka pada server GitHub dengan aman, dan memungkinkan juga untuk berkolaborasi dengan pengguna lainnya. Selain itu, GitHub juga memiliki fitur lainnya seperti permintaan fitur pada repositori, *bug tracking*, manajemen pekerjaan, integrasi berkelanjutan/*continuous integration*, dan kontrol akses untuk membatasi siapa saja yang dapat berkolaborasi beserta cakupan hak aksesnya terhadap sebuah repositori (Uzayr, 2022).

Penelitian ini akan mempergunakan GitHub sebagai penyedia layanan *hosting* Git, untuk menyimpan serta mempublikasikan paket pustaka *user provider* ini melalui Packagist.

2.9 Laravel

Laravel adalah kerangka kerja (*framework*) aplikasi web berbasis bahasa pemrograman PHP yang mengadopsi fitur-fitur terbaik dari solusi kerangka kerja aplikasi web lain, beberapa diantara kerangka kerja yang dimaksud adalah *Ruby on Rails* dan *ASP.NET* (Gilmore, 2018). Sebagai kerangka kerja, Laravel menerapkan sintaks kode yang ekspresif dan elegan, serta menyediakan struktur dan titik mula untuk membuat aplikasi; mengarahkan pengembang web untuk fokus menciptakan suatu karya yang hebat, dimana tim pengembang Laravel yang akan mengurus hal detail lain didalam kerangka kerja (Otwell, 2020b).

Terdapat beberapa fitur yang menjadi nilai unggul dari Laravel, antara lain menerapkan pola desain perangkat lunak berbasis *Model-View-Controller* (MVC), konsep pemrograman berorientasi objek/*object oriented programming* (OOP), sistem pengemasan paket modular dengan manajer dependensi khusus, berbagai cara untuk mengakses basis data relasional, utilitas yang membantu dalam penerapan dan pemeliharaan aplikasi, *templating engine* untuk menampilkan halaman web dinamis menggunakan Blade, dan penulisan baris kode pemrograman yang berorientasi ke *syntactic sugar* (Bean, 2015).

Sebagai kerangka kerja dengan fitur yang komplit, Laravel cocok digunakan untuk berbagai macam proyek aplikasi web, baik dari aplikasi web sederhana, hingga proyek besar level *enterprise* (Korop, 2018).

Perjalanan Laravel dimulai pada tahun 2011, dimana Taylor Otwell selaku pencetusnya merilis versi 1.0 dari Laravel pada bulan Juni 2011. Versi ini belum benar-benar menggunakan arsitektur MVC. Arsitektur ini baru mulai berfungsi lengkap pada perilisan Laravel 2.0. Pada rilis Laravel 3.0, mulai diperkenalkan *command line* Artisan dan dukungan tertanam untuk basis data relasional. Sedangkan pada rilisan Laravel 4.0, seluruh kode sumbernya ditulis ulang dan dirilis dengan nama “Illuminate” (Surguy, 2013). Dimulai dari versi 4.0 ini, hingga versi terbaru saat penelitian ini dilakukan, yaitu versi 8.x, Laravel menggunakan

Composer sebagai manajer paket untuk meng-*install* dependensi yang dibutuhkan.

Rilisan stabil terbaru dari Laravel saat penelitian ini dilakukan adalah versi 8.x, yang dirilis pada 8 September 2020; rilisan ini mensyaratkan penggunaan bahasa pemrograman PHP dengan versi minimal 7.3. Rilisan ini memiliki dukungan *bugfix* sampai dengan 26 Juli 2022 dan dukungan perbaikan celah keamanan sampai 24 Januari 2023 (Otwell, 2020c).

2.9.1 Laravel Breeze

Laravel Breeze adalah pustaka pihak pertama, dikembangkan oleh para pengembang *framework* Laravel, yang berkutat di permasalahan autentikasi. Laravel Breeze mengimplementasikan fitur autentikasi yang yang tersedia pada framework Laravel dan umum ditemui pada aplikasi web, seperti *login*, register, *reset password*, dan konfirmasi alamat email. Laravel Breeze mempergunakan beberapa teknologi penampilan antarmuka yang dapat dipilih berdasarkan preferensi pengembang; apakah ingin tetap bersama Blade yang tradisional, atau menggunakan *library* JavaScript seperti React atau Vue yang lebih modern, dengan Inertia.js sebagai jembatan penghubungnya (Otwell, 2021d).

Dalam penelitian ini, Laravel Breeze akan diinstal bersama-sama dengan *framework* Laravel yang menjadi bahan penelitian ini, dikarenakan fungsi dan antarmuka untuk fitur autentikasi *default* Laravel berada dalam paket pustaka ini, mengakibatkan paket pustaka ini harus diinstal bersamaan dengan *framework* Laravel yang akan diteliti.

2.10 *Middleware*

Middleware merupakan perangkat lunak, yang menjembatani celah antara aplikasi, peralatan, dan basis data dengan tujuan untuk memberikan layanan yang seragam terhadap pengguna (Talend.com, 2021). *Middleware* memungkinkan untuk menciptakan komunikasi, layanan aplikasi, perpesanan, autentikasi, manajemen API, dan manajemen data yang lebih baik antara beragam jenis aplikasi yang membantu proses pertukaran data (Banger, 2020). *Middleware* mengizinkan definisi antarmuka yang jelas untuk server, dimana klien dapat memanggil atau mengaksesnya dengan mudah (Etzkorn, 2017).

Menurut **Banger** (2020), *middleware* terbagi dalam beberapa jenis, antara

lain:

1. *Middleware* basis data (*database middleware*)
2. *Remote Procedure Call* (RPC)
3. *Middleware* Objek
4. Server Aplikasi Web
5. *Message Oriented Middleware* (MOM)
6. Portal
7. *Embedded Middleware*
8. *Application Programming Interface* (API)
9. *Content Centric Middleware*

Dalam penerapannya di arsitektur *framework* web seperti Laravel, istilah *middleware* ini umumnya mengacu pada ke komponen perangkat lunak, yang berada dalam urutan tertentu pada jalur pipa (*pipeline*) pemrosesan permintaan dan respons pesan HTTP, serta berfungsi untuk menangani pekerjaan khusus, seperti koneksi dan akses ke basis data (MDN Web Docs, 2021a).

2.11 User Provider

User provider pada skema autentikasi yang berlaku dalam *framework* Laravel, mendefinisikan mengenai bagaimana data pengguna diterima dari penyimpanan persisten (tetap/permanen). Secara *default*, *framework* Laravel menyediakan *user provider* yang dapat digunakan untuk mengautentikasi pengguna menggunakan basis data, baik melalui *query builder* maupun model Eloquent (Otwell, 2020a). Dalam penelitian ini, penulis akan mengembangkan *user provider*, yang dapat mengautentikasi data pengguna ke layanan REST API, sesuai kebutuhan pada sistem yang berjalan di Papon.id.

2.12 Rekayasa Terbalik (*Reverse Engineering*)

Terdapat beberapa definisi mengenai topik rekayasa terbalik atau *reverse engineering* ini, antara lain:

1. Rekayasa terbalik adalah sebuah proses dimana perangkat lunak, mesin, pesawat terbang, struktur arsitektur dan produk lainnya dipecah, atau dibongkar, untuk menemukan informasi rancangan terkait produk

yang diteliti. Umumnya, rekayasa ini melibatkan proses pembongkaran komponen penyusun dari sebuah produk besar (Hess, 2019).

2. Rekayasa terbalik merupakan sebuah konsep untuk mempelajari alur kerja suatu perangkat lunak aplikasi. Konsep ini bertujuan untuk mengetahui dan mendapatkan celah atau kelemahan dari aplikasi yang dimaksud (Risyan, 2020).
3. Rekayasa terbalik atau rekayasa mundur adalah prosedur dan proses dalam membongkar objek untuk mengetahui bahan, cara kerja, atau teknologi yang dipakai, demi memperoleh informasi mengenai struktur kerjanya yang menyebabkan objek tersebut dapat bekerja dengan baik (Tech, 2019).

Dari beberapa definisi diatas, dapat kita tarik kesimpulan bahwa rekayasa terbalik (*reverse engineering*), adalah satu upaya atau teknik, yang dilakukan dengan cara memecah atau membongkar satu objek utuh, misalnya perangkat lunak, menjadi komponen penyusun yang lebih kecil, misalnya fungsi spesifik dari perangkat lunak, dengan tujuan untuk mempelajari, mencari kelemahan atau celah, dan membuat ulang komponen penyusunnya ketika komponen aslinya tidak tersedia lagi untuk dimodifikasi atau dikembangkan lebih lanjut.

Dalam penelitian ini, penulis menggunakan metode rekayasa terbalik untuk mempelajari struktur autentikasi pada *framework* Laravel, yaitu dengan cara membongkar dan mempelajari kode sumber dari *framework* Laravel, dan kemudian membuat permodelan diagram kelas dan diagram aksi UML, untuk menjelaskan lebih lanjut mengenai cara kerja autentikasi pada Laravel. Diharapkan bahwa metode ini mampu memberi petunjuk mengenai hal-hal yang telah disebutkan sebelumnya.

2.13 *Unified Modelling Language* (UML)

Unified Modelling Language (UML) adalah bahasa standar permodelan untuk pengembangan perangkat lunak dan pengembangan sistem. UML dibangun atas beberapa fakta yang terjadi dalam pengembangan sistem, seperti rumitnya perancangan aplikasi berskala besar, dimana aplikasi komputer sederhana hingga sistem perusahaan *multi-tier* dapat terdiri dari ratusan, bahkan ribuan komponen

perangkat lunak penyusunnya. Bagaimana tiap anggota tim pengembang dapat melacak pemakaian komponennya? Bagaimana cara anggota tim berbagi pakai komponen yang ada? Bagaimana cara tim untuk menyatukan komponen yang masif tersebut? Berkaca dari kerumitan tersebut, maka permodelan sistem, terutama dengan memakai standar UML, menjadi sangat penting dalam pengembangan perangkat lunak (Miles & Hamilton, 2006).

Dalam perancangan sistem, pengembang membuat permodelan untuk satu alasan penting: manajemen kompleksitas. Permodelan membantu pengembang untuk melihat "hutan dari pepohonan" atau poin penting diantara banyak hal yang kurang representatif, memungkinkan untuk lebih fokus mendokumentasikan dan mengomunikasikan aspek penting dari rancangan sistem terkait.

Karena model adalah abstraksi dari benda nyata, permodelan juga membantu membuang detail yang tidak penting, menjadikan model sebagai simplifikasi dari sistem asli yang akan dibangun. Kesederhanaan ini yang menjadikan pemodelan menggunakan UML memiliki nilai keunggulan tersendiri, Untuk dapat membangun model yang efektif, kita perlu satu hal yang sangat penting: sebuah bahasa yang dimana model itu dapat dijelaskan, dan itulah fungsi dari UML (Miles & Hamilton, 2006).

Menurut **Miles & Hamilton** (2006), penggunaan UML dalam permodelan perangkat lunak memiliki beberapa keuntungan dibandingkan dengan tidak memakai UML, seperti:

1. Bahasa yang formal

   UML menggunakan bahasa dengan definisi yang kuat pada setiap elemennya, yang mencegah terjadinya kesalahpahaman akan sesuatu pada tiap model atau bagian sistem yang dibangun diatasnya.
2. Ringkas

   Keseluruhan bahasa ini dibuat dengan notasi sederhana dan ringkas, memudahkan pembaca untuk memahami maksud pembuatnya.
3. Komprehensif

   UML membahas keseluruhan aspek penting dalam sebuah sistem.
4. Dapat diskalakan

UML dapat diandalkan untuk menggambarkan rancangan sistem dari segala bentuk, baik yang sederhana sampai ke sistem yang besar dan masif.

5. Dibangun dari pengalaman

   UML merupakan kumpulan praktik terbaik (*best practice*) dari komunitas penggiat *open source*, yang diracik dari pengalaman selama 15 tahun terakhir sebelum penerbitan buku ini (Learning UML 2.0).

6. Merupakan standar permodelan

   UML sendiri diatur oleh kelompok standar terbuka (*open standards*) dengan kontribusi aktif oleh vendor perangkat lunak dan akademisi di berbagai penjuru dunia, sehingga mencegah terjadinya "*vendor lock-in*", dan memungkinkan standar UML ini dapat digunakan dengan baik, tanpa harus terikat pada vendor atau produk tertentu.

UML terdiri dari berbagai macam *view model*, dimana tiap *view model* menggambarkan tiap perspektif tertentu dari bagian sistem yang dirancang. *View model* itu antara lain:

1. Gambaran logika/*logical view*

   Gambaran ini mendeskripsikan rincian abstrak mengenai bagaimana sebuah sistem dibuat, dan bagaimana tiap komponen berinteraksi dengan komponen lainnya. Diagram UML yang termasuk dalam kelas gambar ini adalah diagram kelas, objek, *state machine*, dan interaksi (Miles & Hamilton, 2006).

2. Gambaran proses/*process view*

   Gambaran ini menggambarkan proses yang terjadi didalam sistem. Umumnya diagram di kelas ini berguna untuk memvisualisasikan apa yang harus terjadi didalam sistem. Kelas gambar ini umumnya berisi diagram aktivitas (Miles & Hamilton, 2006).

3. Gambaran pengembangan/*development view*

   Gambaran ini mendeskripsikan bagaimana bagian penyusun pada sistem dikelompokkan menjadi modul dan komponen. Gambaran ini berguna sekali untuk mengatur tumpukan lapisan didalam arsitektur

sistem yang dikembangkan. Kelas gambar ini umumnya berisi diagram komponen dan paket (Miles & Hamilton, 2006).

4. Gambaran fisik/*physical view*

   Gambaran ini mendeskripsikan bagaimana sebuah sistem dirancang, seperti yang telah dijelaskan sebelumnya di tiga rancangan sebelumnya, yang dibuat menjadi sesuatu yang nyata sebagai kumpulan entitas dunia nyata. Diagram di kelas ini umumnya berisi diagram *deployment* (Miles & Hamilton, 2006).

5. Gambaran kasus guna/*use case view*

   Gambaran ini mendeskripsikan bagaimana fungsionalitas sebuah sistem dimodelkan dari perspektif dunia luar sistem. Gambaran ini diperlukan untuk menceritakan apa yang seharusnya dilakukan sebuah sistem. Semua gambaran yang lainnya akan bergantung pada gambaran kasus guna ini untuk memandu jalannya kelas gambar lainnya. Gambaran ini biasanya terdiri dari diagram *use case*, deskripsi, dan diagram *overview* (Miles & Hamilton, 2006).

Dari banyaknya jenis diagram yang dimiliki oleh UML, penelitian ini hanya menggunakan *component diagram* untuk menggambarkan kedudukan paket pustaka ini terhadap proyek web berbasis Laravel yang mempergunakannya, *class diagram* untuk menggambarkan hubungan antar kelas dari pustaka *user provider* beserta komponen pendukungnya, dan *sequence diagram* untuk memodelkan alur data dalam proses autentikasi pada *framework* Laravel dengan keterlibatan *user provider* didalamnya.

2.14 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah upaya atau proses mengeksekusi perangkat tersebut, dengan tujuan untuk menemukan masalah. Sekitar 70% waktu yang digunakan dalam proses pengembangan perangkat lunak, dihabiskan dalam tahap pengujian ini. Pengujian ini berbeda dengan *debugging*, dimana proses *debugging* bertujuan untuk menghilangkan masalah atau galat dari perangkat lunak, sedangkan pengujian ini hanya bertujuan untuk menemukan galat yang belum ditemukan. Kita menguji perangkat lunak yang kita bangun dengan masukan yang valid dan tidak

valid, dan membandingkan hasil yang kita harapkan dengan hasil yang dikeluarkan dari pengamatan perilaku perangkat lunak tersebut (Chopra, 2018).

Terdapat 3 (tiga) level pengujian pada proyek pengembangan perangkat lunak yang umum dikenal, yaitu pengujian unit (*unit testing*), pengujian integrasi (*integration testing*), dan pengujian sistem (*system testing*). Pengujian unit umumnya dilakukan oleh pengembang perangkat lunak terkait. Pengujian unit ini digunakan untuk menguji metode dan kelas individual dalam kode program yang dibuat, dimana tentunya hal ini transparan dan diketahui secara jelas oleh pengembangnya. Karena sifatnya yang transparan, maka pengujian unit ini juga dikenal sebagai *white-box testing* alias pengujian kotak putih (Dooley, 2017).

Pengujian integrasi pada umumnya dilakukan oleh organisasi pengujian terpisah, misalnya departemen pengujian terdedikasi pada sebuah tim pengembang perangkat lunak. Fungsi dari pengujian ini adalah menguji interaksi dari kumpulan kelas dan modul yang berinteraksi satu dengan lainnya, melalui serangkaian antarmuka yang mendasari interaksi itu. Penguji melakukan pengujian berdasarkan pengetahuan mereka mengenai antarmuka modul yang akan diuji, tetapi tidak berdasarkan informasi mengenai bagaimana suatu modul dikembangkan dari antarmuka yang tersedia. Oleh karena fakta tersebut, pengujian ini juga disebut sebagai *gray-box testing* alias pengujian kotak abu-abu (Dooley, 2017).

Pengujian sistem seharusnya dilakukan oleh departemen pengujian khusus. Pengujian ini adalah tahap akhir, dimana keseluruhan produk perangkat lunak diuji sebagai satu kesatuan sistem. Pengujian sistem ini melibatkan garis haluan (*baseline*) dari produk perangkat lunak dan garis haluan final yang dicanangkan untuk dirilis ke pelanggan. Departemen pengujian khusus yang menguji produk perangkat lunak, akan menggunakan dokumen spesifikasi kebutuhan perangkat lunak (*software requirement specification*/SRS), dan menulis sendiri rutin pengujian mereka, tanpa mengetahui apapun mengenai bagaimana perangkat lunak tersebut dirancang dan dibangun. Oleh karena hal tersebut, pengujian ini disebut juga sebagai *black-box testing*, atau pengujian kotak hitam, karena penguji tidak mengetahui apapun tentang isi perangkat tersebut, kecuali data masukan dan keluarannya yang dapat diamati (Dooley, 2017).

Dalam penelitian untuk mengembangkan paket pustaka *user provider*

berbasis REST API pada *framework* Laravel, penulis akan menggunakan metode *white-box testing* untuk melakukan pengujian unit (*unit testing*), yaitu dengan cara menulis kode pengujian untuk setiap skenario pengujian, karena pengujian ini bertujuan untuk menguji baris kode dari fitur yang dikembangkan, dengan cara membuat kasus uji sesuai dengan alur proses yang terjadi di sistem saat melakukan proses yang mempergunakan pustaka *user provider* ini.

Pressman dan Maxim (Pressman & Maxim, 2020) menyatakan bahwa dikarenakan hakikat dari pengujian *white box* yang bersifat transparan terhadap struktur kode sumber dari komponen yang diujikan, maka kasus uji yang dibuat untuk menguji komponen terkait haruslah menjamin bahwa:

1. Setiap jalur independen didalam modul harus diuji setidaknya sekali.
2. Menguji semua kondisi logika yang dihasilkan dari tiap percabangan.
3. Menguji setiap perulangan dalam batasannya.
4. Menguji struktur data internal terhadap validitas data.

Maka, berdasarkan ketentuan tersebut, metode pengujian *white box* dirasa cocok digunakan dalam pengujian paket pustaka *user provider* yang akan dikembangkan ini.

2.14.1 Mekanisme Pengujian Dalam *Framework* Laravel

Framework Laravel memiliki mekanisme pengujian bawaan, dimana pengujian yang disediakan didalam *framework* tersebut menggunakan PHPUnit sebagai *runtime* untuk menjalankan pengujian unit (*unit testing*) dan pengujian fitur (*feature testing*). Dalam instalasi standar proyek Laravel, sudah terdapat folder `tests`, dengan subfolder `Feature` dan `Unit`, dimana folder `Feature` adalah tempat menyimpan pengujian fitur, dan folder `Unit` sebagai tempat meletakkan pengujian unit. Secara umum, pengujian unit hanya menguji bagian yang sangat kecil dan terisolasi dari kode yang ditulis, sedangkan pengujian fitur menguji bagian yang lebih luas dari aplikasi berbasis Laravel, termasuk mengenai bagaimana objek saling bekerja sama, atau bahkan permintaan HTTP lengkap ke sebuah *endpoint* (Otwell, 2020d).

Untuk menulis kode pengujian, cukup melakukan eksekusi perintah berikut di dalam folder proyek yang diinginkan:

```
php artisan make:test NamaTest
```

Perintah diatas akan membuat sebuah berkas baru bernama **NamaTest.php** didalam folder `tests/Feature`. Jika ingin membuat kode pengujian unit, maka cukup tambahkan *flag* `--unit`, sehingga perintahnya akan menjadi:

```
php artisan make:test NamaTest --unit
```

Dalam pengembangan paket pustaka, rutin pengujian fitur dan unit bawaan *framework* Laravel tidak tersedia untuk digunakan, dikarenakan saat pengembangan paket pustaka, lingkungan dan fungsi pengujian dari Laravel tidak tersedia. Untuk mengatasi hal ini, dibutuhkan sebuah *helper* penguji bernama Testbench (Zaki, 2022), dimana pustaka ini akan menyediakan simulasi lingkungan Laravel beserta *runtime* pengujian yang digunakan, sehingga pengujian unit dan fitur untuk paket pustaka khusus dapat dilakukan walaupun berada diluar lingkungan *framework* Laravel.