BAB III
PELAKSANAAN KERJA PROFESI

3.1 Bidang Kerja

Pada pelaksanaan Kerja Profesi (KP) di PT. Bumi Rejeki Agung, praktikan berprofesi sebagai *Backend Developer* yang bersama menjadi bagian divisi IT dari tim *Web Development*, dalam proses pengerjaannya sendiri praktikan dominan dalam merancang dan membangun sebuah *Application Programming Interface* (API) dengan gaya arsitektur *Representational State Transfer* (REST). Berikut merupakan tugas praktikan selama menjalani Kerja Profesi menjalani tugas antara lain :

a. Merancang bangun API dengan arsitektur REST.
b. Mengimplementasikan rancang bangun API.
c. Melakukan pengujian seluruh konfigurasi API.
d. Memastikan bahwa semua request dan response API dapat berjalan sebagaimana mestinya.
e. Turut berpartisipasi bekerja sama dengan tim lain.

3.2 Pelaksanaan Kerja

Berdasarkan rangkaian bidang kerja diatas pekerjaan utama yang dilakukan praktikan meliputi tahap, yaitu :

3.2.1 Analisa Sistem

Dalam tahap analisis sistem, dimana praktikan melakukan identifikasi masalah dan memahami mekanisme sistem sebagai landasan perencanaan sistem sesuai dengan kebutuhan dan apa yang akan direncanakan:

1. Wawancara

   Wawancara diperlukan untuk menyampaikan pendapat dan ide agar dapat memperoleh informasi berdasarkan kebutuhan. Dengan melakukan wawancara kepada direktur mengenai apa saja kebutuhan sistem yang diperlukan agar memudahkan praktikan menganalisis

dan mendesain sistem yang nantinya akan dirancang seusai dan selaras dengan apa yang dibutuhkan.

2. Studi Literatur

   Pada tahap ini praktikan melakukan pengkajian literatur terkait permasalahan yang ada, seperti metode pengembangan sistem berbasis API menggunakan arsitektur REST, Laravel, JSON, MySQL. Literatur yang direview adalah jurnal akademik, artikel, buku referensi, dan serta situs web lainnya.

3. Identifikasi Aktor

   Mengidentifikasi aktor bertujuan untuk mengetahui pengguna dan merupakan salah satu langkah pertama dalam melakukan analisis setiap entitias yang terlibat dalam sebuah sistem dan memberikan gambaran apa yang dikerjakan oleh sistem yang disajikan pada **Tabel 3.1**.

Tabel 3.1 Identifikasi Aktor

Aktor	Deskripsi
Admin	Pihak yang mengelola data properti dan data reservasi
Pengunjung	Pengguna yang melakukan input reservasi hunian

4. Identifikasi Data

   Kebutuhan data apa saja yang akan digunakan selama melakukan konsumsi API. Data yang di perlukan antara lain:

   a. Data Properties

      Data ini digunakan untuk menampung id, nama properti, tipe property, lokasi property, deskripsi properti, nomor yang dapat dihubungi dengan beberapa file dalam bentuk gambar.

   b. Data Reservation

      Data ini digunakan untuk menampung id, nama, email, nomor telepon pengunjung, model, tipe properti yang dipilih yang saat sedang melakukan reservasi dan catatan waktu saat pengunjung melakukan reservasi.

5. Analisis Kebutuhan Fungsionalitas

   Menganalisa kebutuhan fungsionalitas untuk mengetahui kebutuhan yang berisi proses-proses apa saja yang akan diperlukan oleh sistem. Pada **Tabel 3.2** dibawah merupakan analisis kebutuhan fungsionalitas.

Tabel 3.2 Analisis Kebutuhan Fungsionalitas

Aktor	Deskripsi
Admin	Mengelola data konten properti pada web meliputi lihat, tambah, ubah dan hapus
Admin	Melakukan *approve* saat terdapat reservasi dari pengunjung
Admin	Mengelola data reservasi meliputi lihat dan hapus
Pengunjung	Pengguna yang dapat melakukan reservasi mengenai properti

6. Analisis Kebutuhan Data

   Proses menentukan dan mendokumentasikan data yang dibutuhkan guna menunjang kebutuhan informasi. Sebagai berikut:

   a. Data Properties

      Data properties yang diperlukan adalah id, properties_name, type, location, image, properties_description, notelp.

   b. Data Reservation

      Data reservation yang diperlukan adalah id, reservation_name, email, notelp, model, type, timestamps().

3.2.2 Perancangan Sistem

Sesudah melakukan tahapan menganalisa sistem, kemudian adalah melakukan rancang sistem, dimana proses rancangan dibentuk. Tahapan membentuk Application Programming Interface (API) dengan gaya arsitektur Representational State Transfer (REST).

1. Desain Alur Proses

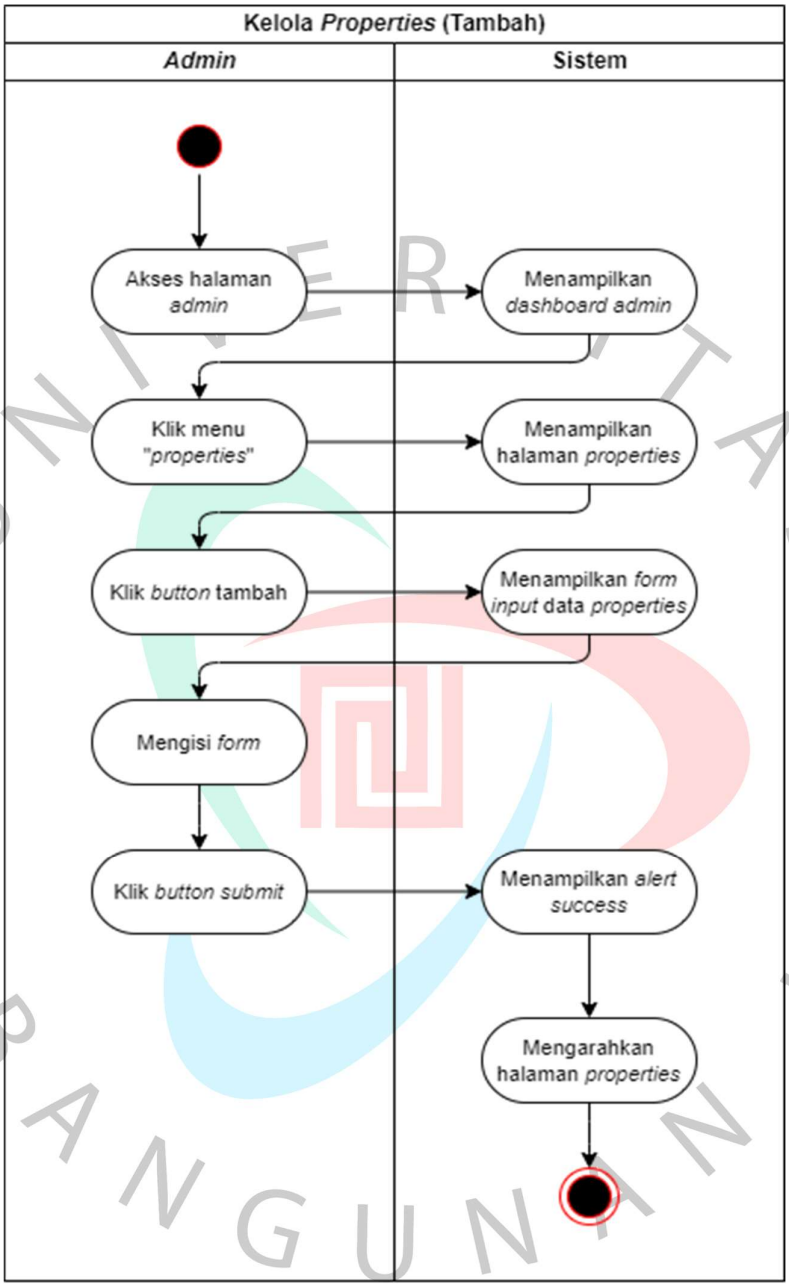
   Desain alur proses in merupakan proses yang sudah diusulkan dan bertujuan untuk memperoleh proses rangkaian cara kerja suatu

sistem yang diterapkan. Desain aliran proses yang sudah diusulkan nantinya desain tersebut akan dianalisa berdasarkan kebutuhan fungsional. Desain aliran proses digambarkan dengan activity diagram. Berikut ini adalah *activity diagram* yang yang sudah diusulkan.

a) *Activity diagram* kelola *properties*

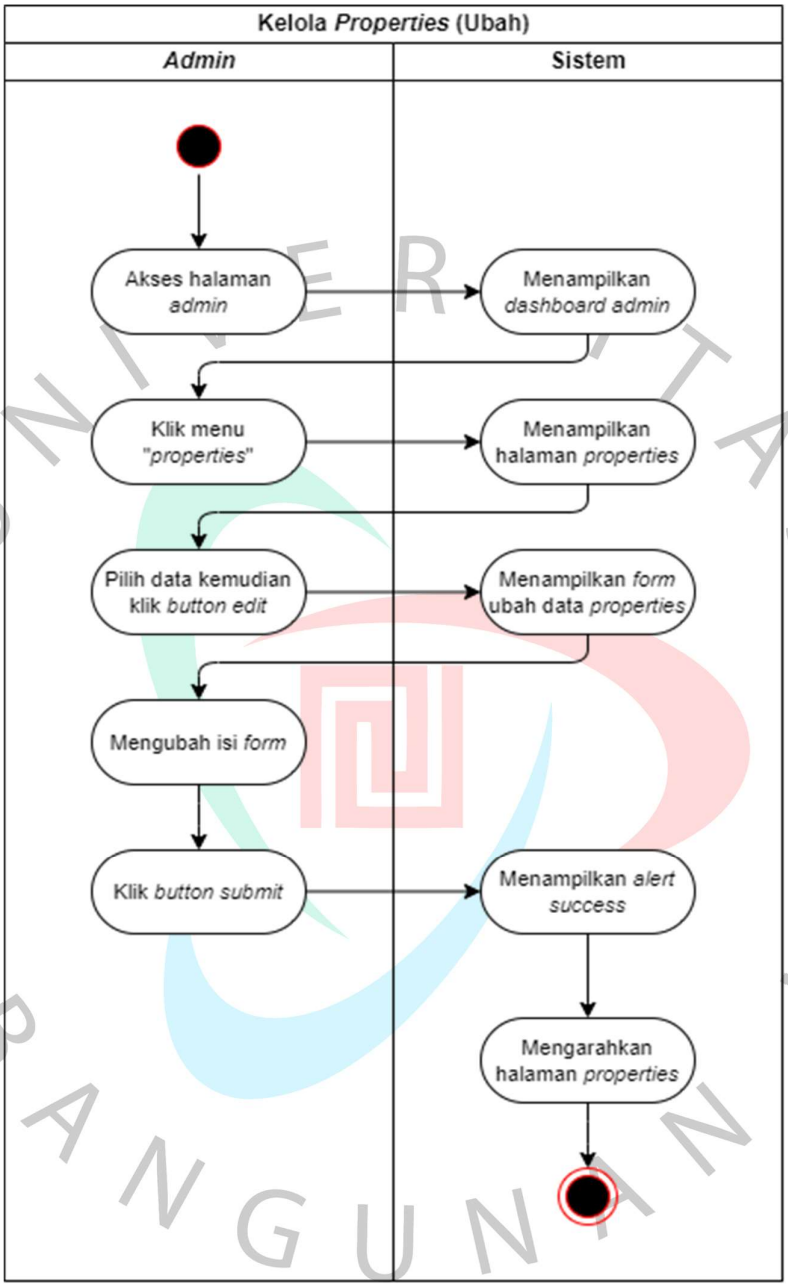
Activity diagram pengelolaan pada data *properties* ini merupakan proses dimana sebuah aktor *admin* mengelola data. Data – data tersebut adalah data yang berisi informasi dari sebuah konten properti. Pengelolaan tersebut dilakukan oleh aktor *admin* yang sudah mengakses halaman utama *admin* setelah itu dapat menampilkan dan menjalankan metode tambah, ubah serta hapus data *properties*. Dibawah ini merupakan seluruh *activity diagram* kelola *properties* yang disajikan pada **Gambar 3.1** untuk tambah data *properties*, **Gambar 3.2** untuk ubah data *properties* dan **Gambar 3.3** untuk hapus data *properties*.

- *Activity diagram* tambah *properties*



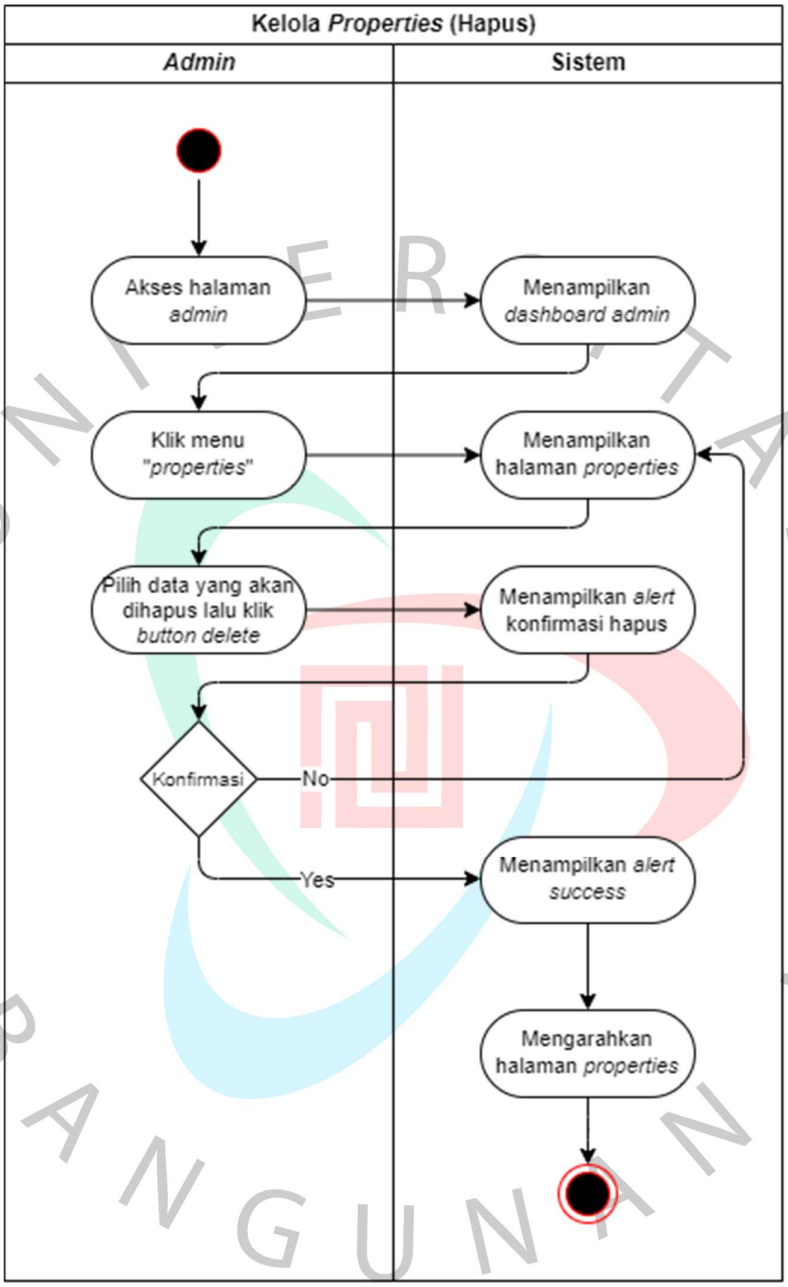
Gambar 3.1 *Activity Diagram* Tambah Data *Properties*
Sumber: Dokumentasi Praktikan

- *Activity diagram* ubah *properties*



Gambar 3.2 *Activity Diagram* Ubah Data *Properties*
Sumber: Dokumentasi Praktikan

- *Activity diagram* hapus *properties*

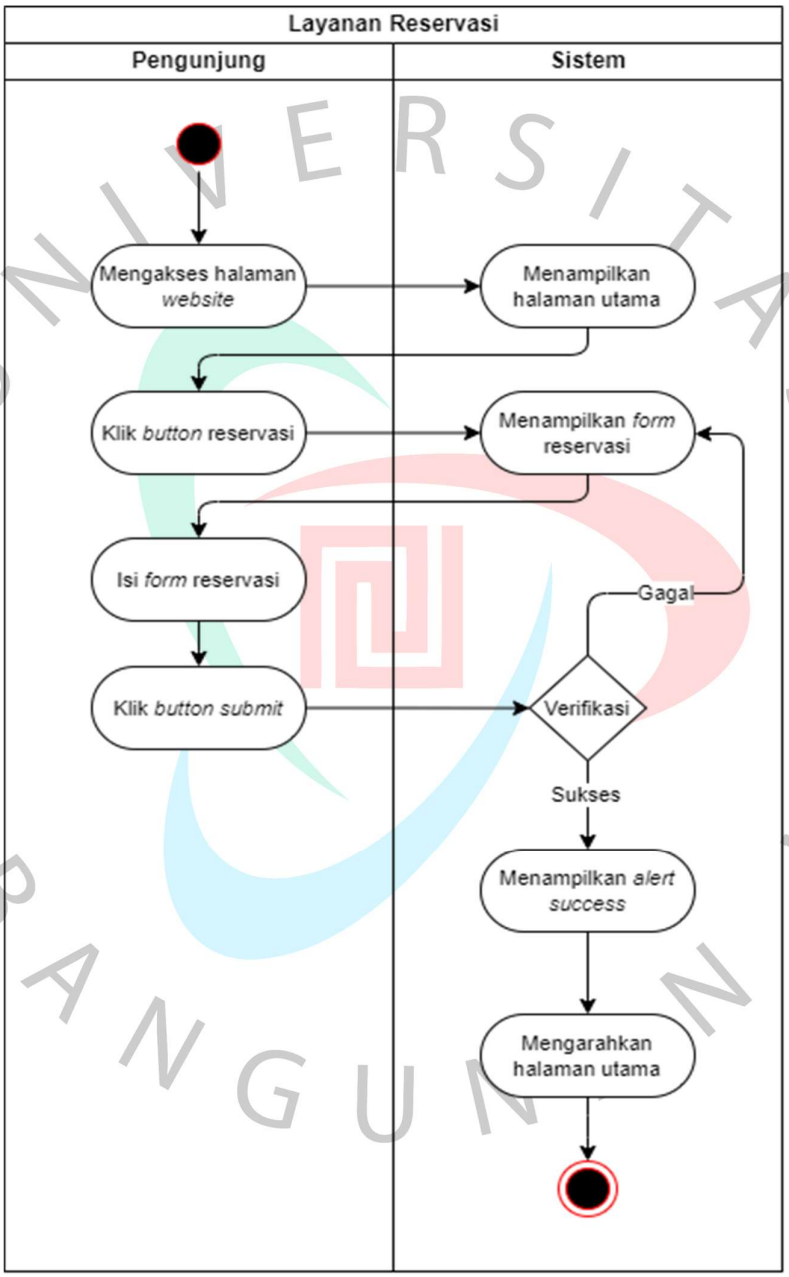


Gambar 3.3 *Activity Diagram* Hapus Data *Properties*
Sumber: Dokumentasi Praktikan

b) *Activity diagram* layanan reservasi

Activity diagram layanan reservasi disajikan pada **Gambar 3.4** yang merupakan proses dimana pengunjung melakukan pengisian reservasi dalam *form input* dengan data berupa nama, *email*,

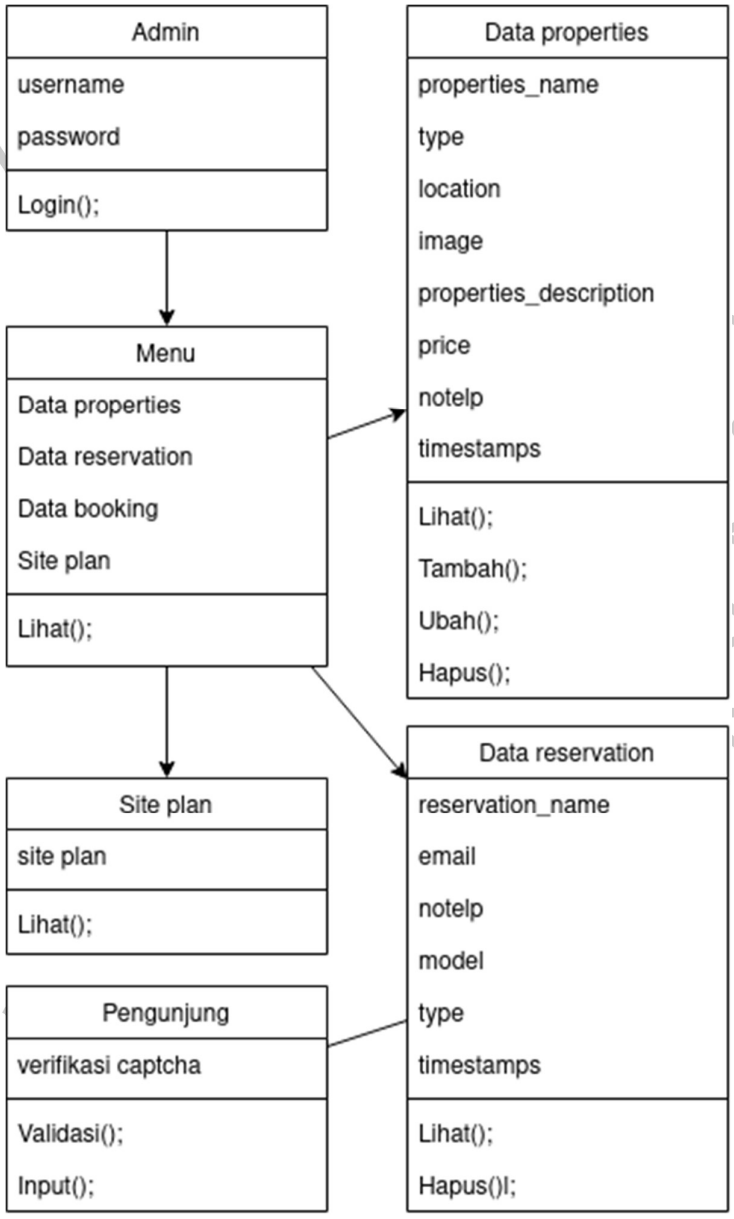
nomor telepon, *model* dan tipe yang dipilih oleh pengunjung reservasi yang nantinya dibentuk menjadi data *reservation* dengan *key* reservation_name, email, notelp, model dan type



Gambar 3.4 *Activity Diagram* Layanan Reservasi
Sumber: Dokumentasi Praktikan

c) *Class Diagram*

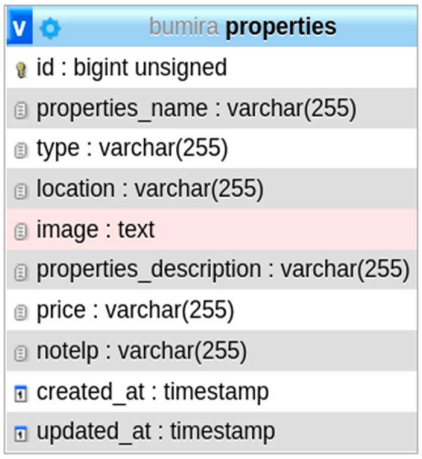
Praktiklan melakukan visualisasi struktur kelas dari sistem dan menunjukkan hubungan antar kelas. Berikut adalah class diagram yang dianalisa dapat dilihat pada **Gambar 3.5**.



Gambar 3.5 *Class Diagram*
Sumber: Dokumentasi Praktikan

d) Desain *Database*

Desain *database* merupakan desain dari tabel data *properties* dan tabel data *reservations* yang akan menjadi fokus dan acuan dalam pembuatan sistem API. MySQL adalah *Database Management System* (DBMS) yang berjalan dengan struktur *Structured Query Language* (SQL) dalam penyimpanan data yang digunakan untuk membuat skema data. Berikut skema *database* yang akan terfokus pada data *table properties* (**Gambar 3.6**) dan *table reservations* (**Gambar 3.7**):



Gambar 3.6 Tabel Data *Properties*
Sumber: Dokumentasi Praktikan

Gambar 3.7 Tabel Data *Reservations*
Sumber: Dokumentasi Praktikan

e) Desain *Application Programming Interface* (API)

Desain API akan menentukan fungsionalitas yang mendasarinya dari penerapan API dengan arsitektur REST berdasarkan hasil analisis sistem. Desain ini dimaksudkan untuk fokus pada pengembangan sistem agar tetap berada dalam lingkup pengembangan. Untuk menyederhanakan desain API, dibuatlah model desain API *properties* pada **Tabel 3.3** dan model desain API *reservation* pada **Tabel 3.4** sebagai berikut:API

Tabel 3.3 Model API Data *Properties*

Method	Path	Keterangan
GET	`/api/properties`	Menampilkan seluruh data properties
POST	`/api/properties`	Menambah data properties
PUT	`/api/properties/{$id}`	Mengubah data berdasarkan id data properties
DELETE	`/api/properties/{$id}`	Menghapus data berdasarkan id data properties

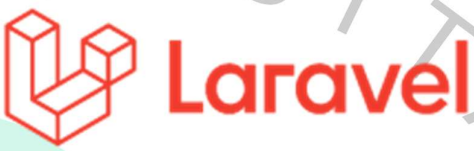
Tabel 3.4 Model API Data *Reservation*

Method	Path	Keterangan
GET	`/api/reservation`	Menampilkan seluruh data reservasi
POST	`/api/reservation`	Mengirim data reservasi
PUT	`/api/reservation/{$id}`	Mengubah data berdasarkan id data reservasi
DELETE	`/api/reservation/{$id}`	Menghapus data berdasarkan id data reservasi

3.2.3 Pendukung Sistem

a) Spesifikasi dan *tools* yang digunakan

Setelah desain sistem maka dilakukanlah implementasi sistem. *Application Programming Interface* (API) yang diimplementasikan sesuai kebutuhan yang sudah di rancang dengan arsitektur REST. Berikut adalah spesifikasi dan tools dalam membangun API:



Gambar 3.8 Logo Laravel

1. Laravel

   Laravel merupakan *framework* berbais PHP dan juga bersifat *open-source*. Laravel dapat memungkinkan pemrogram untuk membangun API dengan gaya arsitektur REST dengan menggunakan struktur MVC atau dikenal dengan *Model*, *View* dan *Controller* yang setiap komponennya dapat membantu pemrogram fokus pada satu aspek pengembangan pada satu waktu dengan membagi proses pembuatan API menjadi tiga bagian: *Model*, *View*, dan *Controller*.

   a. *Model*

      Komponen ini terkait dengan *database* dan mewakili data yang sedang ditransfer antara komponen *Controller*.

   b. *View*

      Komponen *View* digunakan untuk menyajikan data kedalam tampilan interaksi dengan meminta *Model* untuk memberikan informasi.

   c. *Controller*

      Komponen dimana seluruh fungsional dibangun yang nantinya dapat menangani *request* HTTP yang masuk dan mengirim *response* kembali menuju *client*.

2. MySQL

   MySQL merupakan *Database Management System* (DBMS), yang digunakan sebagai alat manajemen database untuk mengelola struktur dalam *database* sisi *server* menggunakan bahasa SQL.

3. POSTMan

   Postman adalah aplikasi yang bertindak sebagai REST *Client* untuk menguji REST API dengan permintaan HTTP untuk mendapatkan dan memvalidasi berbagai jenis tanggapan. Postman sering digunakan sebagai alat untuk menguji API yang ditulis oleh pengembang API.

4. JavaScript Object Notation (JSON)

   JavaScript Object Notation (JSON) adalah format teks baku untuk merepresentasikan data terstruktur berdasarkan sintaksis objek JavaScript. JSON. Biasanya JSON digunakan untuk menampilkan keluaran data yang dihasilkan.

3.2.4 Implementasi & Pengujian Sistem

a) Pengkodean

Pada fase ini desain perancangan diimplementasikan kedalam bahasa pemrograman dan dilakukan juga uji coba pada API arsitektur REST. Implementasikan API menggunakan *framework* Laravel berbasis bahasa pemrograman PHP. *Controller* adalah bagian penting dari pemrograman MVC yang bertindak sebagai penghubung antara *view* dan *model*. Di dalam pengontrol ada banyak logika pemrograman untuk menempatkan fungsionalitas tertentu. Pada umunya berbagai pemrosesan dilakukan di dalam *controller*. Berikut proses pada pengkodean API dengan gaya arsitektur REST:

1. *Route*

   Route merupakan daftar baris seluruh perintah yang dapat digunakan untuk semua rute yang terdaftar pada API. Perintah - perintah ini menampilkan metode, *path*, *resource* dan *controller* untuk *route* yang

disertakan pada **Gambar 3.9** dan penjelasan route disajikan kedalam bentuk **Tabel 3.5**.

Gambar 3.9 Baris Perintah Route
Sumber: Dokumentasi Praktikan

Tabel 3.5 Penjelasan *Route*

Method	URI	*Route*	*Controller Class*	Keterangan
GET	`api/resource`	`resource.index`	`Api/ResourceController@index`	Menampilkan seluruh data
POST	`api/resource`	`resource.store`	`Api/ResourceController@store`	Menambahkan data
GET	`api/resource/{id}`	`resource.show`	`Api/ResourceController@show`	Menampilkan data berdasarkan id
PUT	`api/resource/{id}`	`resource.update`	`Api/ResourceController@update`	Mengubah data berdasarkan id
DELETE	`api/resource/{id}`	`resource.destroy`	`Api/ResourceController@destroy`	Menghapus data berdasarkan id

Dari penjelasan diatas dapat disimpulkan bahwa, sebagai contoh ketika *client* melakukan request dengan `api/properties` menggunakan *method* GET maka *route* yang terpanggil adalah `properties.index` dari *class* didalam PorpertiesController bernama `function index()`, jika *client* menggunakan *method* POST *route* yang digunakan adalah `properties.store` yang merupakan *class* dialam PorpertiesController bernama `function store()`. Untuk pendeklarasian *function* itu sendiri dapat dilihat pada **Gambar 3.10**.

```
public function index() {
// GET
}
public function store() {
// POST
}
public function show() {
// GET
}
public function update() {
// PUT
}
public function destroy() {
// DELETE
}
```

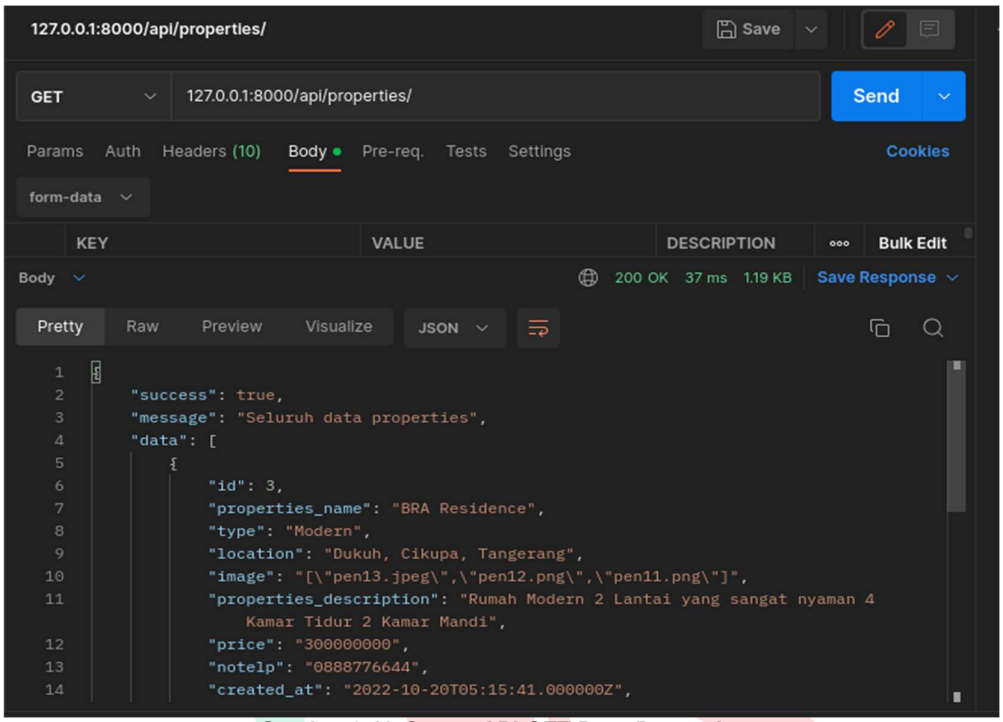
Gambar 3.10 Penggunaan *Function*
Sumber: Dokumentasi Praktikan

```
public function index()
{
      $data = Properties::all();
      return new PropertiesResource(true, 'Seluruh data
      properties', $data);
}
```

Gambar 3.11 *Source Code Properties* GET
Sumber: Dokumentasi Praktikan

2. *Properties* GET

Pada **Gambar 3.11** terdapat baris `Properties::all();` yang digunakan agar dapat menangkap seluruh data *properties* dan `function index()` sebagai *method* GET untuk menampilkan data *properties*.



Gambar 3.12 *Output* API GET Data *Properties*
Sumber: Dokumentasi Praktikan

Output atau keluaran pada **Gambar 3.12** diatas menghasilkan *output* dalam bentuk JSON yang berisi seluruh data *properties* yang ditandai deklarasi array `[]` dan objek `{}` didalamnya "`data":[{}]` dari hasil uji API dengan URI `127.0.0.1:8000/api/properties/` menggunakan *method* GET.

```
public function store(Request $request)
{
      $validator = Validator::make($request->all(), [
            'properties_name' => 'required',
            'type' => 'required',
            'model' => 'required',
            'image' => 'required|array',
            'image.*' =>
            'required|image|mimes:jpg,jpeg,png,gif|max:10240',
            'properties_description' => 'required',
            'price' => 'required',
            'notelp' => 'required',
      ]);

      if ($validator->fails()) {
            return new PropertiesResource(false, 'Data gagal
            ditambahkan!', null);
      }

      $imgName = [];
      foreach ($request->file('image') as $img) {
            $filename = $img->getClientOriginalName();
            $img->move(public_path() . '/images/blueprints', $
            filename);
            $imgName[] = $filename;
      }

      $post = new Properties;
      $post->properties_name = $request->properties_name;
      $post->type = $request->type;
      $post->model = $request->model;
      $post->properties_description = $request
      ->properties_description;
      $post->price = $request->price;
      $post->notelp = $request->notelp;
      $post->image = json_encode($imgName);

      $post->save();
      return new PropertiesResource(true, 'Data berhasil
      ditambahkan.', $post);
}
```

Gambar 3.13 *Source Code Properties* POST
Sumber: Dokumentasi Praktikan

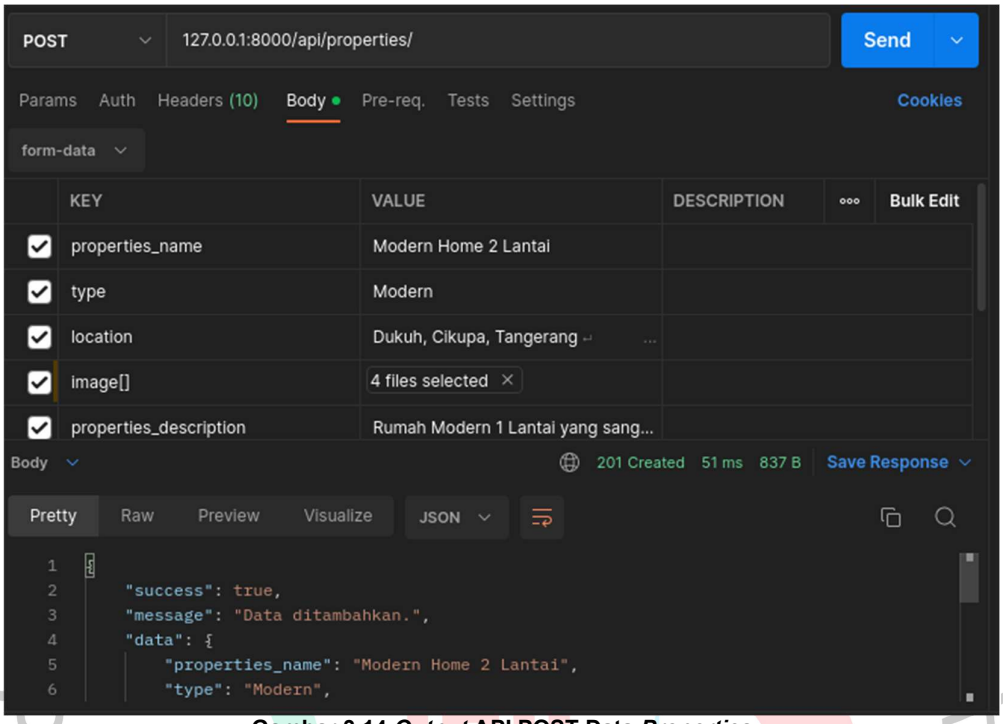
3. Data *Properties* POST

Function `store()` pada **Gambar 3.13** diatas adalah untuk menambahkan data dengan menagkap data yang sesuai masukan *input* serta keterangan diikuti dengan nama variable, `store()` digunakan sebagai *method* POST data *properties*. Sebelum proses data, dilakukanlah validasi terlebih dahulu yang disajikan pada **Tabel 3.6** dibawah, berikut adalah keterangannya.

Tabel 3.6 Validasi POST Data *Properties*

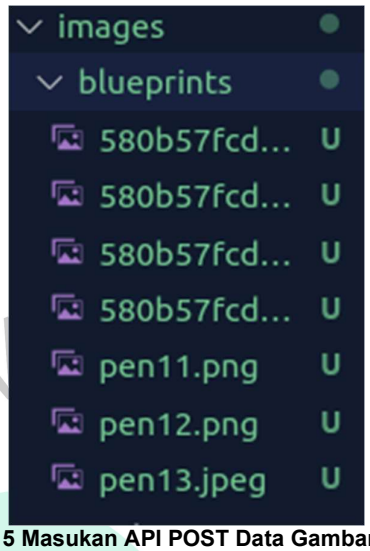
Key	**Validasi**	**Keterangan**
properties_name	required	*Field* wajib diisi
type	required	*Field* wajib diisi
location	required	*Field* wajib diisi
	required | array	*Field* wajib diisi dan data bisa *array*
	image	*Field* harus berupa gambar.
image	mimes:jpg,jpeg,png,gif | max:10240	*Field* harus berekstensi jpg, jpeg, png, gif dengan maksimal setiap gambar 1024Kb / 10Mb
properties_description	required	*Field* wajib diisi.
price	required	*Field* wajib diisi.
notelp	required	*Field* wajib diisi.

Jika data yang sudah di*request* lalu dikirimkan tidak sesuai dengan validasi di atas, maka akan dikirim kembali sebagai pesan dalam format JSON dengan kode *false* dengan pesan “Data gagal ditambahkan!”. Namun, jika data yang dikirimkan benar maka data image akan diunggah ke dalam server. Setelah data image berhasil diunggah, langkah selanjutnya yaitu memasukkan data ke dalam *database* dan menampilkan pesan *true* “Data berhasil ditambahkan”.

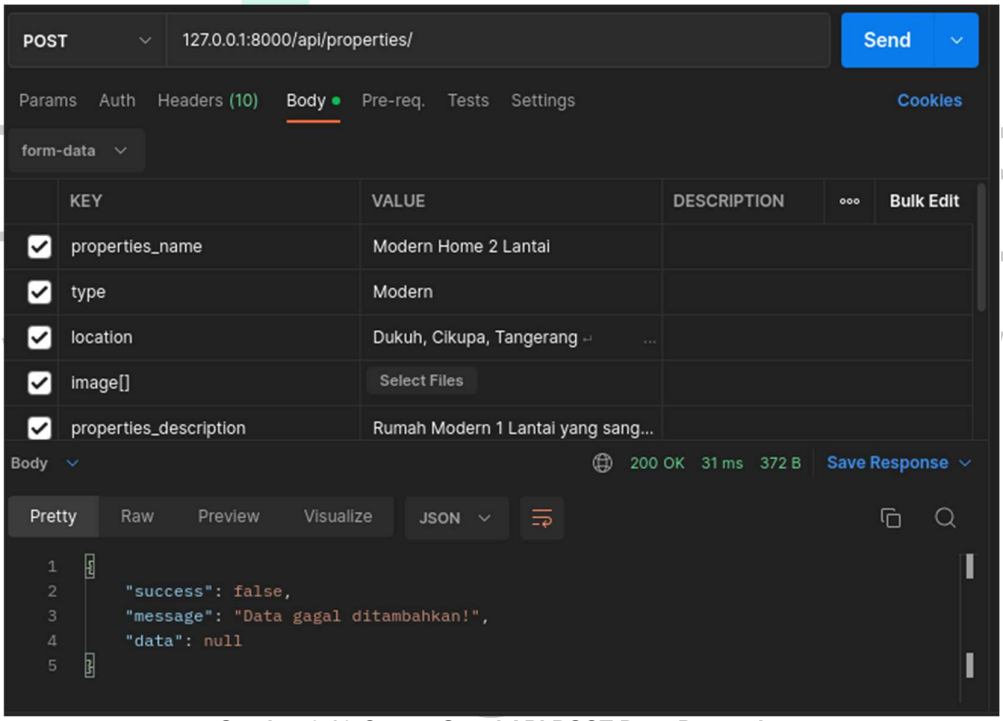


Gambar 3.14 *Output* API POST Data *Properties*
Sumber: Dokumentasi Praktikan

Pada **Gambar 3.14** diatas merupakan pengujian API POST data *properties*. *Output* diatas menghasilkan pesan *true* dengan *message* "Data ditambahkan" yang artinya semua *form* sudah tervalidasi dengan benar, untuk *output image* dapat dilihat pada **Gambar 3.15** yang bernama *file* "580b57fcd…" berjumlah 4 (empat) *file*.



Gambar 3.15 Masukan API POST Data Gambar *Properties*
Sumber: Dokumentasi Praktikan



Gambar 3.16 *Output* Gagal API POST Data *Properties*
Sumber: Dokumentasi Praktikan

Jika data gagal divalidasi maka akan menghasilkan pesan *false* dengan *message* “Data gagal ditambahkan” yang artinya *form* tidak tervalidasi dengan benar, contoh kasus diatas pada **Gambar 3.16**

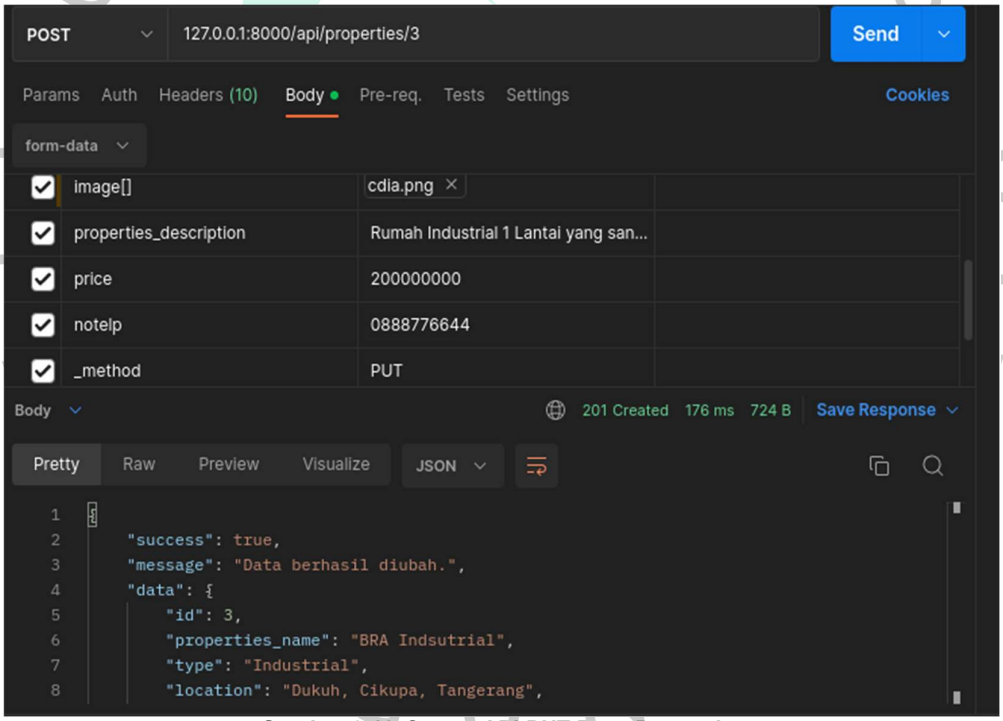
adalah *key* image[] dengan *value* yang tidak terisi sehingga data tersebut gagal untuk disimpan atau data *null*.

```
public function update(Request $request, Properties
$properties, $id)
    {
        $validator = Validator::make($request->all(), [

            // validasi

        ]);

        if ($validator->fails()) {
            return new PropertiesResource(false, 'Data gagal
            diubah!', null);
        }

        $properties = Properties::find($id);
        if ($request->hasFile('image')) {
            if ($oldImg = json_decode($properties->image)) {
                $j = count($oldImg);
                for ($i = 0; $i < $j; $i++) {
                  unlink(public_path('/images/blueprints/' .
                  $oldImg[$i]));
                }
                $properties->delete();
            }
            $imgName = [];
            foreach ($request->file('image') as $img) {
                  $filename = $img->getClientOriginalName();
                  $img->move(public_path() .
                  '/images/blueprints', $filename);
                  $imgName[] = $filename;
            }
        }

        // Simpan request

        $properties->save();
        return new PropertiesResource(true, 'Data berhasil
diubah.', $properties);
    }
```

Gambar 3.17 *Source Code Properties* PUT
Sumber: Dokumentasi Praktikan

4. Data *Properties* PUT

Fungsi baris `update()` pada **Gambar 3.17** diatas sebagai *method* PUT data *properties* yang dapat digunakan untuk memperbarui data sesuai *id* yang ada. Validasi `update()` baris diatas sama dengan *method* `store()` data *properties* pada validasi POST. Untuk pengujiannya sendiri untuk *method* PUT data *properties* maka ditambahkan *form - data* berisi *key* _method dan *value* PUT. Sesuai validasi, jika data yang dikirimkan sudah benar, selanjutnya akan mengecek permintaan *file* gambar. Ketika gambar diminta, maka gambar baru akan disimpan dan menghapus gambar yang lama dari *server* berserta pesan "Data berhasil diubah".

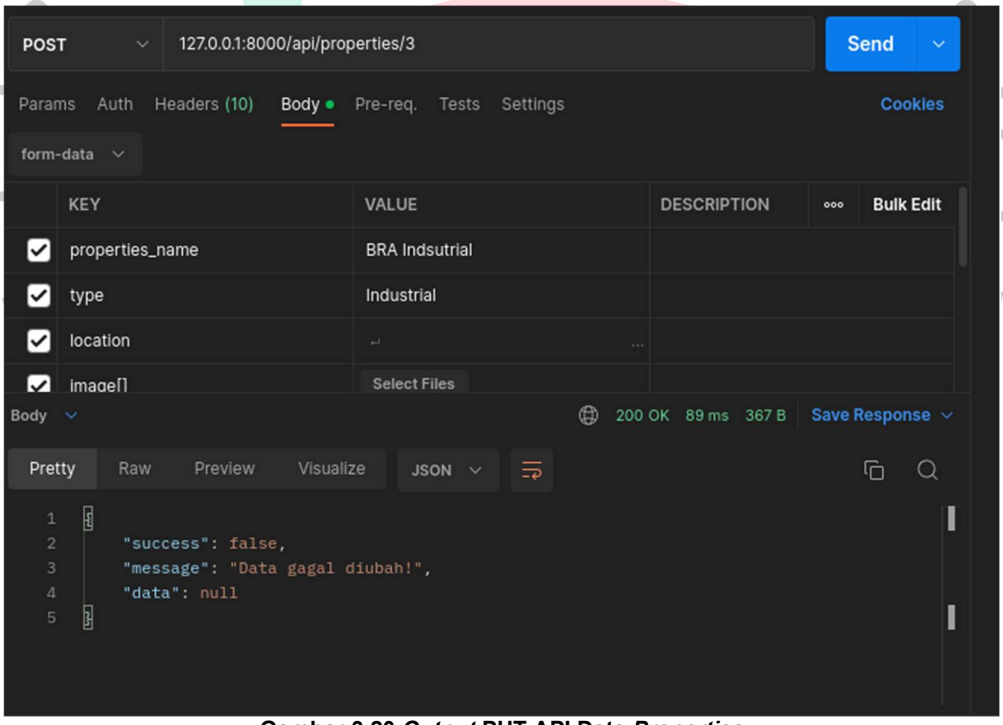


Gambar 3.18 *Output* API PUT Data *Properties*
Sumber: Dokumentasi Praktikan

Output atau keluaran pada **Gambar 3.18** diatas menghasilkan output dalam bentuk JSON yang berisi data *properties* dengan id 3 yang berhasil diubah dari hasil uji API dengan URI `127.0.0.1:8000/api/properties/3` menggunakan metode PUT. Output image dapat dilihat pada **Gambar 3.19** dibawah yang

sudah mengubah data image lama yaitu pen11.png, pen12.png dan pen13.jpeg menjadi cdia.png.

Gambar 3.19 Masukan PUT API Data Gambar *Properties*
Sumber: Dokumentasi Praktikan



Gambar 3.20 *Output* PUT API Data *Properties*
Sumber: Dokumentasi Praktikan

Uji coba gagal **Gambar 3.20** diatas terjadi karena data pada *key* image dan location belum dimasukkan maka data tidak tervalidasi

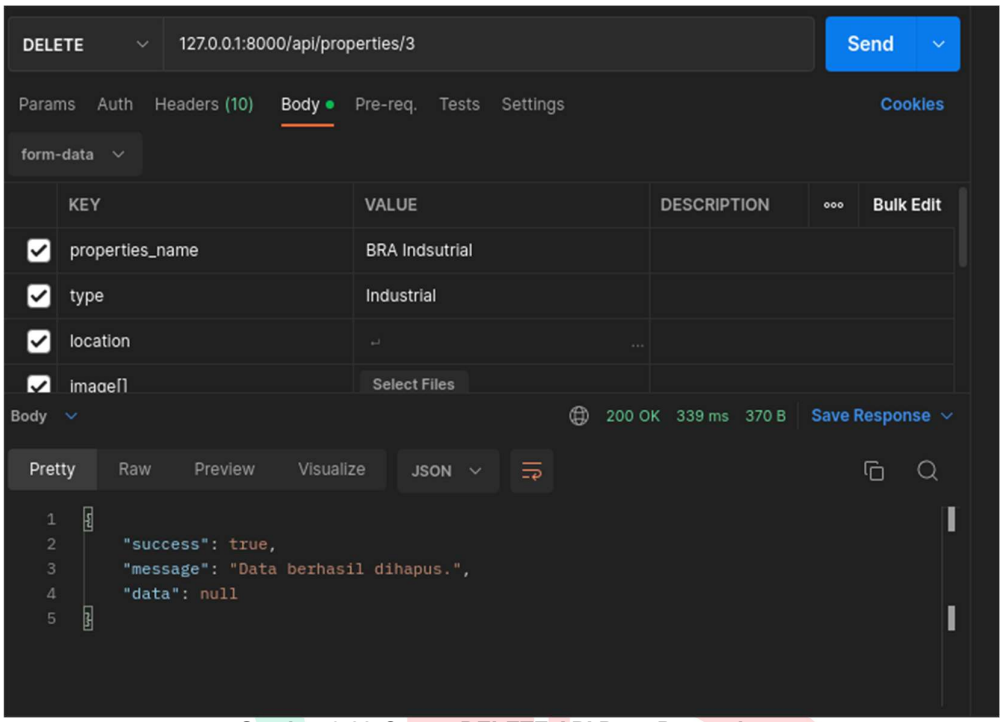
dengan benar data yang dihasilkan null atau gagal tersimpan dengan pesan “Data gagal diubah!”.

```
public function destroy(Properties $properties, $id)
    {
        $properties = Properties::find($id);
        $image = json_decode($properties->image);
        $length = count($image);
        for ($i = 0; $i < $length; $i++) {
            unlink(public_path('/images/blueprints/' .
            $image[$i]));
        }
        $properties->delete();
            return new PropertiesResource(true, 'Data berhasil
            dihapus.', null);
    }
```

Gambar 3.21 *Source Code Properties* DELETE
Sumber: Dokumentasi Praktikan

5. Data *Properties* DELETE

Dari baris kode **Gambar 3.21** di atas, fungsi yang digunakan adalah `destroy()` untuk melakukan *method* DELETE pada data *properties*. Pertama - tama gambar dihapus dari *server* yang terkait dengan data. Setelah gambar berhasil dihapus dari *server*, maka seluruh *field* data properties juga dihapus dari *database*.



Gambar 3.22 *Output* DELETE API Data *Properties*
Sumber: Dokumentasi Praktikan

Output pada **Gambar 3.22** diatas menampilkan data JSON dengan *success true* dengan pesan 'Data berhasil dihapus'. *Method* yang digunakan adalah DELETE dengan URI `127.0.0.1:8000/api/properties/3` yang menghapus data dengan id 3

6. Data *Reservation*

   Sama seperti *method* yang digunakan pada data *properties*. Struktur kode data *reservation* kurang lebih memiliki sintaks seperti `index()`, `store()` dan `destroy()` yang sama dalam menjalankan *method* GET, POST, DELETE. Berikut source code layanan reservasi pada **Gambar 3.23** dibawah.

```
public function index() //GET
{
      Reservastion::all();
}

Public function store(Request $request) //POST
{
      Reservation::create($request->all());
}

Public function destroy(Reservation $reservation)
{
      $reservation->delete();
}
```

Gambar 3.23 *Source Code* Layanan Reservasi
Sumber: Dokumentasi Praktikan

7. Pembahasan

   Pada tahap pembahasan, praktikan memaparkan hasil implementasi *Application Programming Interface* (API) menggunakan gaya arsitektur *Representational State Transfer* (REST) dalam sistem.

   a. *Representational State Transfer* (REST)

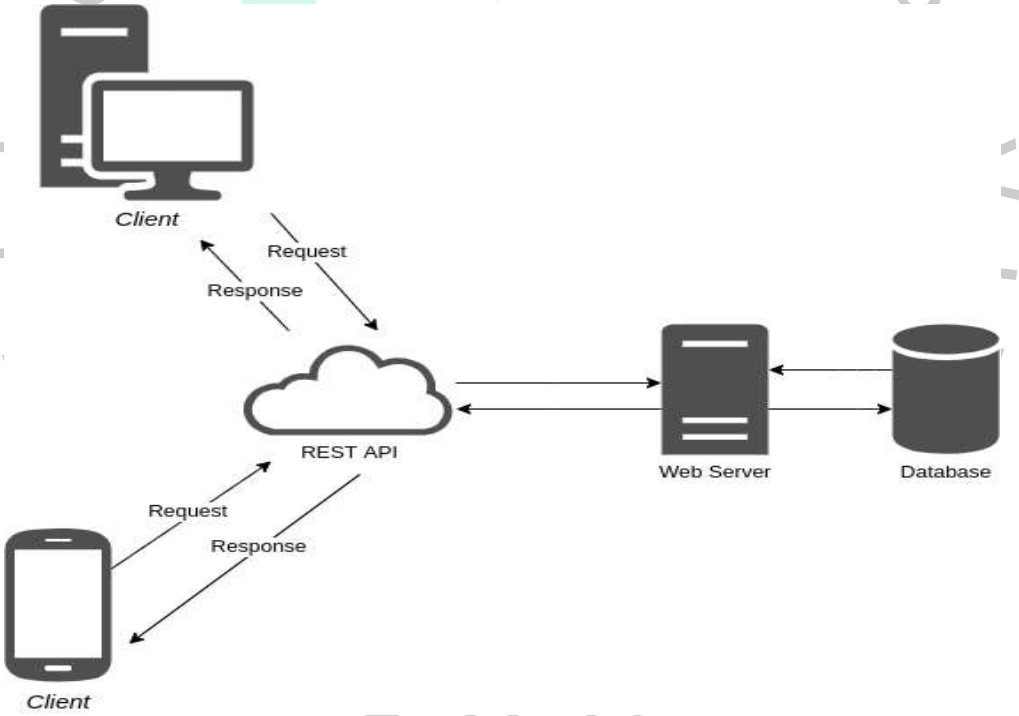
      Dalam menentukan metode *transfer* data API yang baik, praktikan menggunakan gaya arsitektur REST dimana memanfaatkan metode HTTP untuk mengartikan suatu perintah API, yaitu GET, POST, PUT dan DELETE. Dan berikut adalah daftar rute pada **Tabel 3.7** yang dikelola oleh *controller* yang dibuat dengan *resource*.

Tabel 3.7 *Route* Berdasarkan *Resource*

Method	URI	Action
GET	/api/properties	index
POST	/api/properties	store
PUT	/api/properties/{$id}	update
DELETE	/api/properties/{$id}	destroy

GET	/api/reservation	index
POST	/api/reservation	store
DELETE	/api/reservation/{$id}	destroy

Untuk penggunaan API itu sendiri. API akan bertindak sebagai perantara antara *client* dengan *server*, dimana *client* yang melakukan permintaan pada aplikasi, API akan berkerja melaksanakan permintaan tersebut dan meneruskannya menuju server kemudian *server* akan memberikan hasil keluaran tersebut menuju *client* melalui API berupa permintaan sesuai *client*. Untuk lebih jelasnya dapat dilihat pada **Gambar 3.24** dibawah.



Gambar 3.24 Proses Kerja REST API
Sumber: Dokumentasi Praktikan

b. *Uniform Resource Locator* (URL)

Saat menentukan nama URL di API, praktikan membuat beberapa konvensi penamaan. Ada berbagai aturan yang membantu membangun sistem API yang baik.

- `/properties` memilih seluruh data properties
- `/properties/{$id}` memilih data *properties* berdasarkan *id*
- `/reservation` memilih seluruh data *properties*
- `/reservation/{$id}` memilih data *reservation* berdasarkan *id*

c. HTTP *Status Code*

HTTP *status code* adalah pesan *status response* yang dikirim oleh *server* untuk dapat mengenali permintaan dari API. Kode *status response* HTTP yang menunjukkan apakah permintaan HTTP tertentu berhasil diselesaikan. Pesan *status* yang disematkan terdapat di *header* HTTP halaman untuk memberi tahu browser hasil permintaan dan *response* balik.

1. 200 *OK*
   Perintah berhasil saat permintaan benar ke *server*
2. 201 *Created*
   Perintah yang dikirim sukses dan membuat resource baru
3. 400 *Bad Request*
   Perintah yang dikirm salah karena permintaan tidak valid
4. 401 *Unauthorized*
   Autentikasi tidak sah sehinggah *client* gagal mendapatkan *resource* yang dituju
5. 403 *Forbidden*
   Client tidak mempunyai hak akses untuk mendapatkan *response* kedalam *resource* yang dituju
6. 404 *Not Found*
   Resource tidak ditemukan
7. 409 *Conflict*
   Permintaan tidak dapat diproses karena terjadi konflik
8. 500 *Internal Server Error*
   Server mengalami kondisi tak terduga kedalam *resource*

d. *Error Messages*

Pesan kesalahan ini membantu pengembang mengidentifikasi dan mengenali kesalahan seperti parameter yang hilang, kesalahan validasi parameter, dan kesalahan *server*. Di bawah ini adalah baris kode contoh pesan kesalahan digunakan dalam pengembangan API dapat dilihat pada **Gambar 3.25** dibawah.

```
return [
            'success'   => $this->status,
            'message'   => $this->message,
            'data'      => $this->resource
      ];

//Penggunaan message
(Boolean, 'Pesan status', $data);
return new PropertiesResource(true,  'Data berhasil
ditambahkan.', $properties);
```

Gambar 3.25 Penggunaan Pesan
Sumber: Dokumentasi Praktikan

3.3 Kendala Yang Dihadapi

Kendala merupakan kegiatan yang menghambat untu melakukan apa yang seharusnya dilakukan. Dalam melaksanakan kegiatan biasanya kita tidak luput dari hambatan - hambatan yang menghadang. Dalam keadaan apapun, kendala sering kali muncul untuk memberikan sebuah pelajaran. Sama seperti halnya kendala yang dihadapi oleh praktikan selama menjalani pelaksanaan Kerja Profesi (KP). Terdapat beberapa kendala yang dihadapi selama masa praktikan melaksanakan Kerja Profesi (KP) pada PT. Bumi Rejeki Agung. Sebagai berikut:

a. Spesifikasi perangkat *device* yang kurang maksimal membuat praktikan tidak dapat mengimplementasi dan uji coba sistem pada *mobile*.
b. Lokasi pelasanaan KP yang ditempuh cukup jauh dan akan melelahkan.

3.4 Cara Mengatasi Kendala

Terlepas dari berbagai kendala yang dihadapai, praktikan terus berupaya meminimalisir kesalahan dan meningkatkan kualitas dengan

bersikap positif. Berikut ini uraian bagaimana praktikan mengatasi hambatan yang ditemuinya dalam melaksanakan KP:

a. Melakukan diskusi bersama tim dengan mencari, memberikan atau mengusulkan saran - saran dan saling bertukar pikiran untuk mengatasi permasalahan yang ada. Selain itu, melakukan meeting daring bersama tim maupun secara langsung secara berkala.
b. Menggunakan tools alternatif untuk uji coba.
c. Bersikap positif dan professional.
d. Turut membantu semua peran tim.
e. Menginstall sistem operasi yang lebih ringan.

3.5 Pembelajaran Yang Diperoleh dari Kerja Profesi

Pembelajaran yang diperoleh selama praktikan melaksanakan Kerja Profesi (KP) di PT. Bumi Rejeki Agung. Praktikan dapat meningkatkan kemampuan dalam pemrograman dan analisis terutama bisa mengimplementasikan struktur *Application Programming Languange* (API) dengan gaya arsitektur *Representational State Transfer* (REST) dan dapat memahami konsep tersebut. Selain itu praktik harus bisa bekerja secara profesional dengan penuh tanggung jawab untuk menyelesaikan masalah yang ada.