RESTful API

Pendahuluan

Pada praktikum kali ini anda akan mempelajari tentang konsep dari RESTful API, autentikasi token dan proses pembangunan RESTful API.

Tujuan Pembelajaran

1. Mahasiswa memahami konsep RESTful API
2. Mahasiswa mampu membangun autentikasi token pada RESTful API
3. Mahasiswa mampu membangun CRUD dengan RESTful aPI

Alat dan Bahan

1. PC atau Laptop
2. Text Editor/IDE (direkomendasikan PHPStorm atau VSCode)
3. Web Browser
4. PHP
5. Postman

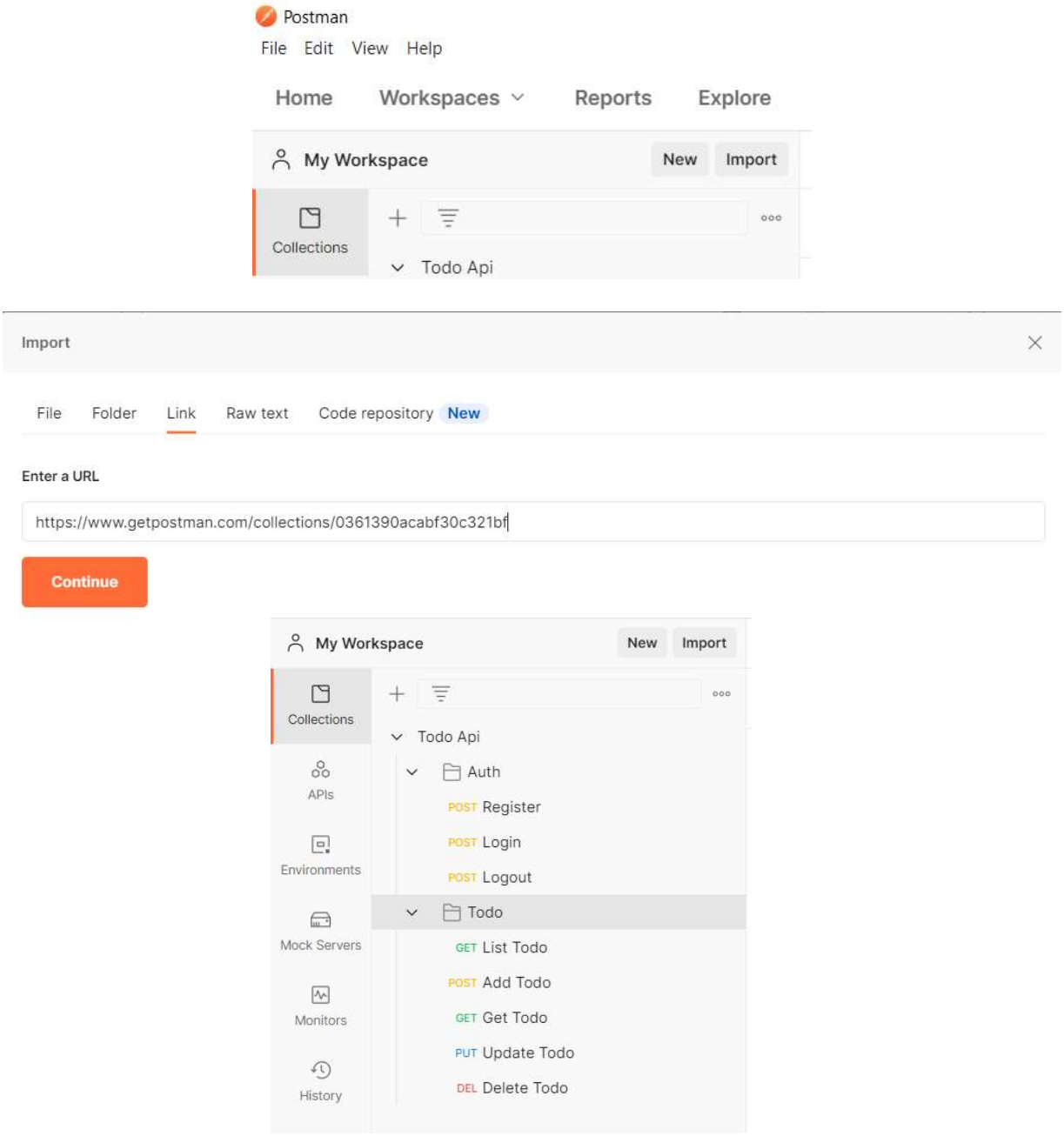
RESTful API

RESTful API adalah salah satu arsitektur dalam API (Application Program Interface) yang menggunakan request HTTP untuk mengakses data. Data diakses dengan menggunakan HTTP method GET, PUT, POST dan DELETE yang merujuk pada operasi pembacaan, pembaruan, pembuatan dan penghapusan pada resource. Selain HTTP method, dalam RESTful atau REST digunakan juga HTTP response untuk mendefinisikan respon data yang dikembalikan. Silahkan membuka tautan berikut untuk mendapatkan informasi terkait HTTP status https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Format respon yang umum digunakan berupa JSON (Javascript Object Notation).

Autentikasi pada RESTful API sedikit berbeda dengan web. Pada RESTful API tidak dikenal adanya session. Umumnya digunakan token untuk menggantikan session. Token ini biasanya disisipkan pada header request HTTP. Dalam framework Laravel sudah mendukung untuk pembangunan API dan juga pembuatan token. Anda bisa menggunakan Laravel Sanctum untuk memudahkan dalam pembangunan RESTful API.

Sebelum masuk ke dalam praktikum, silahkan siapkan beberapa kebutuhan berikut:

1. Postman (https://www.postman.com/downloads/)
2. Import collection https://www.getpostman.com/collections/0361390acabf30c321bf.
3. Cara import collection, buka postman, dan klik menu import di sebelah kiri. Kemudian masukkan link pada poin no. 2, seperti tampilan berikut:



Praktikum 1: Membuat autentikasi token pada RESTful API

Pada praktikum ini, akan dibuat autentikasi token pada RESTful API dengan memanfaatkan Laravel Sanctum.

1. Buatlah proyek baru Laravel dengan menjalankan perintah laravel installer.

```
composer create-project --prefer-dist laravel/laravel=8.* todolist-app
```

2. Tunggu sampai proses instalasi composer selesai dilakukan. Setup proyek menjadi git repository.
3. Tambahkan package laravel sanctum dengan menjalankan perintah berikut

```
composer require laravel/sanctum
```

4. Publish konfigurasi dari Laravel sanctum dengan menjalankan perintah berikut. Proses ini memungkinkan konfigurasi sanctum sesuai kebutuhan dan juga digunakan untuk mendapatkan skema migration.

```
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
```

5. Buka file User.php yang terdapat dalam models. Sisipkan trait HasApiTokens ke dalam file User.php. Trait ini digunakan untuk menambahkan fungsi pembuatan serta penghapusan token.

```php
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
```

6. Buat database kosong baru dengan nama todolistDB menggunakan phpmyadmin atau mysql client.
7. Atur koneksi ke database baru dengan menyesuaikan file `.env` yang ada dalam proyek Laravel.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=todolistDB
DB_USERNAME=root
DB_PASSWORD=
```

8. Jalankan migrate data dengan menjalankan perintah artisan

```
php artisan migrate
```

9. Pastikan proyek bisa dijalankan dan tidak ada kesalahan, dengan menjalankan perintah artisan serve. Lakukan commit untuk langkah setup awal proyek Laravel.

```
php artisan serve
```

10. Untuk memudahkan pembangunan response dari API yang dibangun, tambahkan trait `ApiResponse.php` di lokasi directory `app/Traits/ApiResponse.php`.

ApiResponse.php ×

app > Traits > ApiResponse.php > ...

```php
<?php

namespace App\Traits;

trait ApiResponse
{
    protected function apiSuccess($data, $code = 200, $message = null)
    {
        return response()->json([
            'data' => $data,
            'message' => $message,
        ], $code);
    }

    protected function apiError($errors, $code, $message = null)
    {
        return response()->json([
            'errors' => $errors,
            'message' => $message,
        ], $code);
    }
}
```

11. Selain itu buatlah juga custom request untuk menangani request spesifik terhadap API. Jalankan perintah berikut.

```
php artisan make:request ApiRequest
```

12. Ubah kode pada app/Http/Requests/ApiRequest.php menjadi seperti berikut. Kode berikut mengganti perilaku default dari Laravel jika terjadi kesalahan pada request berupa API.

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use App\Traits\ApiResponse;
use Illuminate\Contracts\Validation\Validator;
use Illuminate\Http\Exceptions\HttpResponseException;
use Illuminate\Http\Response;


abstract class ApiRequest extends FormRequest
{
    use ApiResponse;

    abstract public function rules();

    protected function failedValidation(Validator $validator)
    {
        throw new HttpResponseException($this->apiError(
            $validator->errors(),
            Response::HTTP_UNPROCESSABLE_ENTITY,
        ));
    }

    protected function failedAuthorization()
    {
        throw new HttpResponseException($this->apiError(
            null,
            Response::HTTP_UNAUTHORIZED
        ));
    }
}
```

13. Buat controller untuk menangani autentikasi dengan perintah artisan. Jalankan perintah berikut.

```
php artisan make:controller Api/AuthController
```

14. Tambahkan Trait ApiResponse ke dalam Api/AuthController. Perhatikan kode berikut sebagai referensi.

```php
class AuthController extends Controller
{

    use ApiResponse;
```

15. Untuk mengakses API, user membutuhkan sebuah akun. Sehingga dibutuhkan proses register. Untuk menghindari controller yang terlalu panjang, untuk proses validasi akan ditangani oleh custom request. Buatlah custom request dan beri nama dengan RegisterRequest. Jalankan perintah berikut untuk membuat custom request. Pemisahan proses validasi di luar controller, bertujuan untuk menerapkan konsep *thin controller.*

```
php artisan make:request RegisterRequest
```

16. Buka file RegisterRequest.php, ubah parent class menjadi ApiRequest. Atur nilai kembalian menjadi true pada function authorize() dan sesuaikan proses validasi berdasar kebutuhan proses registrasi akan. Kode selengkapnya, dapat anda lihat sebagai berikut.

app > Http > Requests > RegisterRequest.php > ...

```php
<?php

namespace App\Http\Requests;

class RegisterRequest extends ApiRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8'
        ];
    }
}
```

17. Pada app/Http/Controllers/Api/AuthController.php, tambahkan function register dengan parameter RegisterRequest. Dalam function ini terdapat logika pembuatan user jika validasi berhasil. Sebagai balasan dari API, dikembalikan nilai berupa data user beserta token. Proses pembuatan token, dapat menggunakan function createToken.

```php
public function register(RegisterRequest $request)
{
    $validated = $request->validated();
    $user = User::create([
        'name' => $validated['name'],
        'email' => $validated['email'],
        'password' => Hash::make($validated['password']),
    ]);

    $token = $user->createToken('auth_token')->plainTextToken;
    return $this->apiSuccess([
        'token' => $token,
        'token_type' => 'Bearer',
        'user' => $user,
    ]);
}
```

18. Untuk mengimplementasikan fitur login, buatlah custom request dengan nama LoginRequest dengan perintah artisan. Modifikasi isi dari LoginRequest.php sehingga menjadi kode seperti berikut. Sebenarnya implementasi API dari fitur registrasi dan login tidak jauh berbeda, yang membedakannya adanya perbedaan format respon dari server.

```
php artisan make:request LoginRequest
```

```php
<?php

namespace App\Http\Requests;

class LoginRequest extends ApiRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'email' =>  'required|string|email',
            'password' => 'required|string|min:8',
        ];
    }
}
```

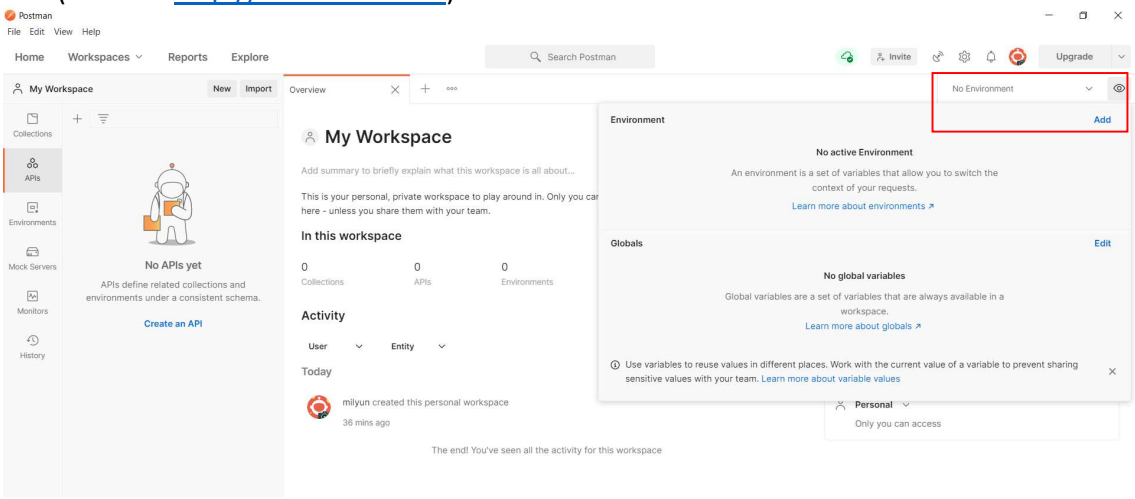
19. Sedangkan implementasi login dapat menggunakan kode berikut. Letakkan function login berikut pada file Api/AuthController.php

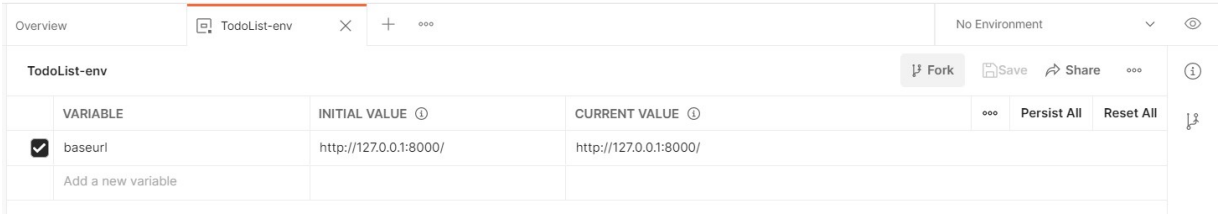
```php
public function login(LoginRequest $request)
{
    $validated = $request->validated();

    if (!Auth::attempt($validated)) {
        return $this->apiError('Credentials not match', Response::HTTP_UNAUTHORIZED);
    }

    $user = User::where('email', $validated['email'])->first();
    $token = $user->createToken('auth_token')->plainTextToken;

    return $this->apiSuccess([
        'token' => $token,
        'token_type' => 'Bearer',
        'user' => $user,
    ]);
}
```

20. Buka file routes/api.php dan tambahkan route baru untuk register dan juga login. Sehingga dalam file routes/api.php menjadi berikut.

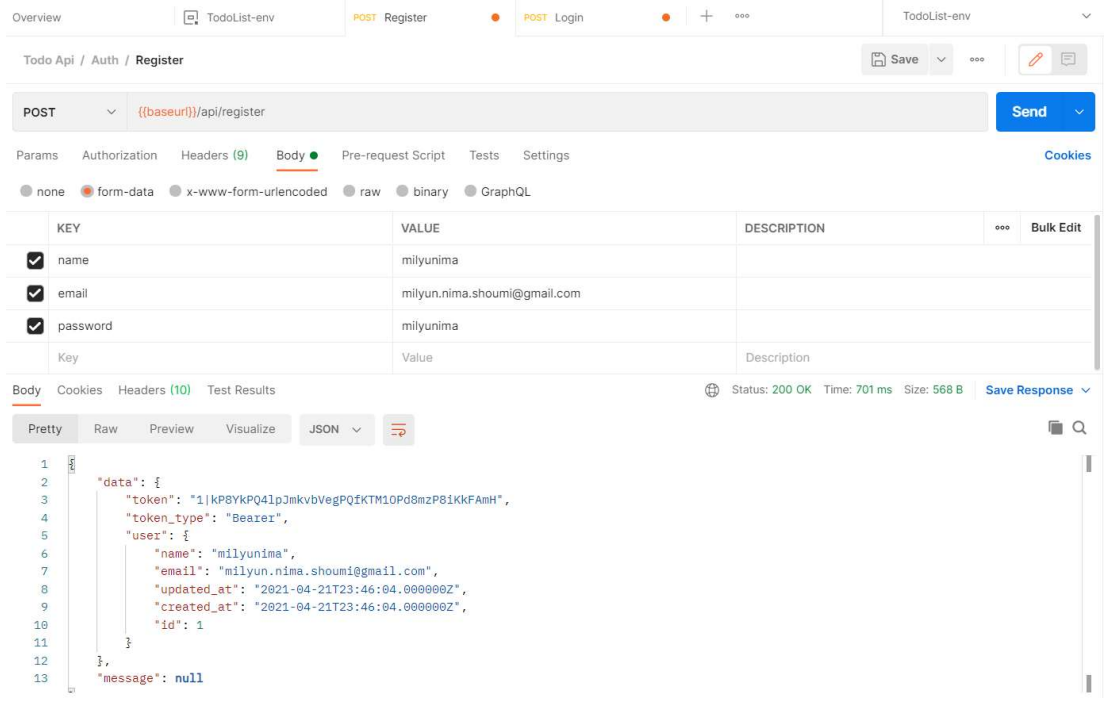
```
Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);
```

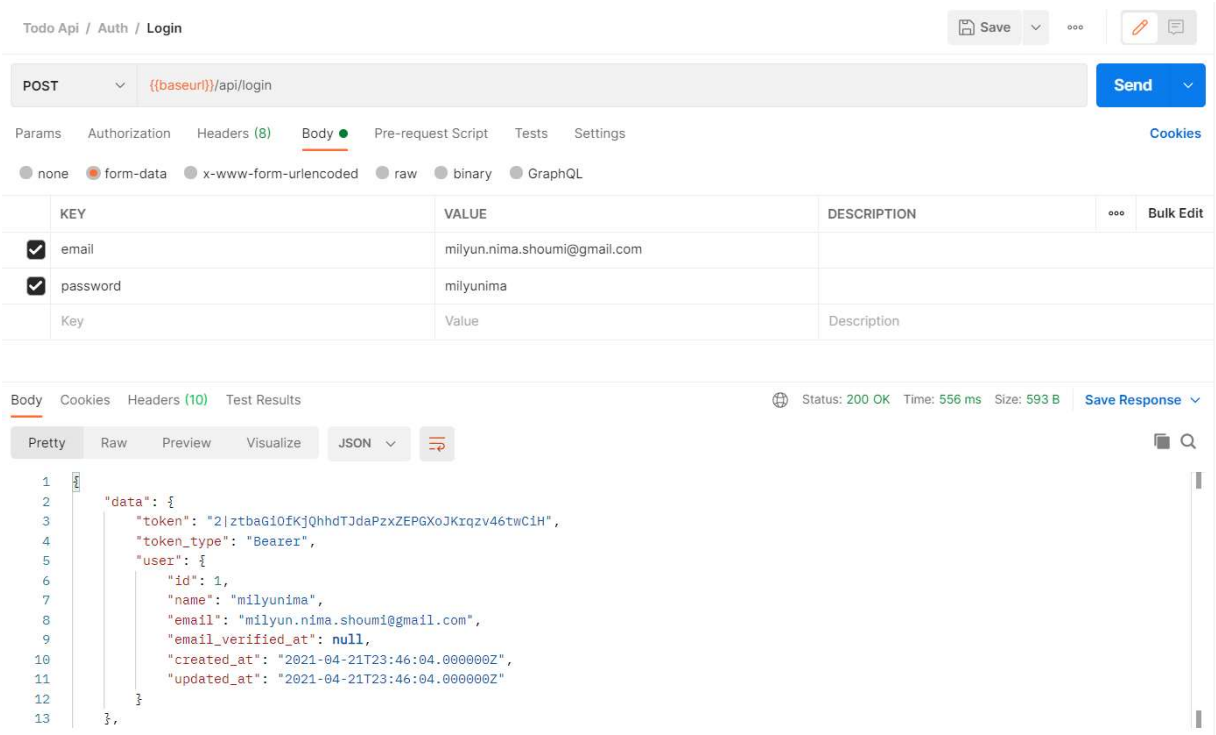
21. Bukalah aplikasi postman, atur env baseurl menjadi path server development anda (Contoh: http://localhost:8000).





22. Uji fungsionalitas fitur register dan juga login. Jika tidak ada kesalahan, anda akan mendapatkan balasan berupa format json yang di dalamnya terdapat token. Lakukan git commit untuk pembuatan fitur register dan login tersebut.





23. Token yang didapatkan digunakan sebagai autentikasi setiap dibutuhkan untuk pemanggilan API. Token ini disisipkan pada header dengan format `Authorization: bearer <token>` (nilai dari <token> akan digantikan dengan token yang didapatkan dari API register atau login).

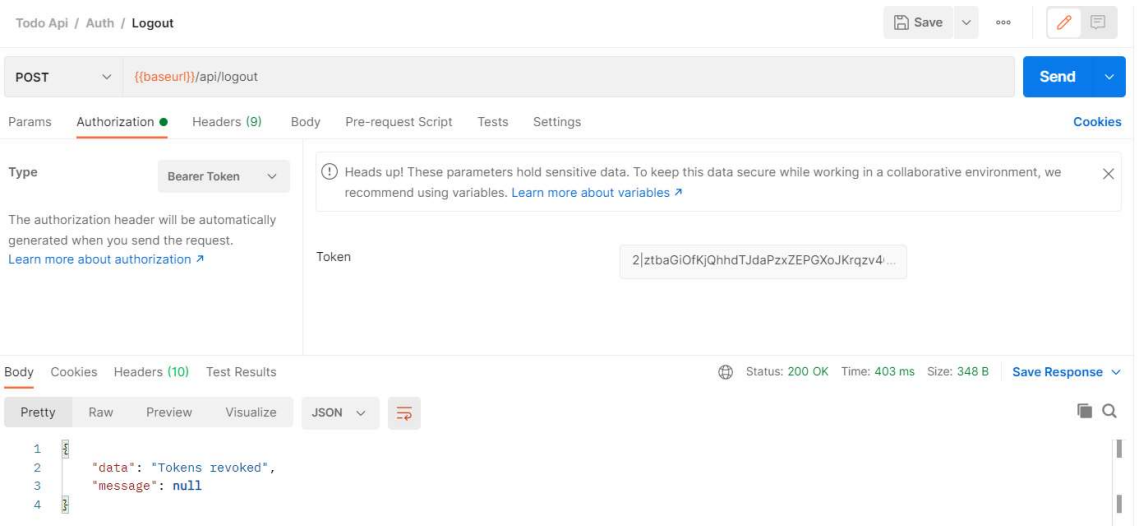
24. Buka kembali file Api/AuthController.php, tambahkan fungsi untuk menghapus token sebagai implementasi fitur logout. Proses logout dilakukan dengan cara menghapus token yang dikirimkan melalui header.

```php
public function logout()
{
    try {
        auth()->user()->tokens()->delete();
        return $this->apiSuccess('Tokens revoked');
    } catch (\Throwable $e) {
        throw new HttpResponseException($this->apiError(
            null,
            Response::HTTP_INTERNAL_SERVER_ERROR,
        ));
    }
}
```

25. Tambahkan route pada routes/api.php untuk endpoint logout. Route logout hanya dapat diakses oleh user yang telah terautentikasi. Sehingga dibutuhkan penambahan middleware auth:sanctum. Perhatikan deklarasi route berikut.

```php
Route::middleware(['auth:sanctum'])->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);
});
```

26. Buka kembali aplikasi postman, lakukan percobaan request logout dengan menyisipkan informasi token pada tab Authorization di postman. Jika anda telah berhasil mendapatkan balasan pesan "`Tokens revoked`" berarti anda telah berhasil menyelesaikan fitur logout.



27. Lakukan git commit untuk implementasi fitur logout sebagai penanda bahwa praktikum 1 telah selesai dilakukan.

Praktikum 2: CRUD dalam RESTful API

Pada praktikum ini, anda akan membangun CRUD dalam API dengan memanfaatkan route apiResource dan controller resource pada Laravel.

1. Buatlah migration untuk mendefinisikan tabel todolist. Tabel ini digunakan untuk menyimpan informasi data todo masing-masing user. Jalankan perintah berikut

```
php artisan make:migration create_todolist_table --table=todolist
```

2. Lengkapi fungsi up pada migration todos dengan kode berikut

```php
public function up()
{
    Schema::create('todolist', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('user_id');
        $table->foreign('user_id')->references('id')->on('users');
        $table->string('todo');
        $table->string('label');
        $table->boolean('done');
        $table->timestamps();
    });
}
```

3. Lengkapi juga fungsi down untuk membalik proses pembuatan tabel todos.

```php
public function down()
{
    Schema::dropIfExists('todolist');
}
```

4. Buatlah controller dan model untuk melayani CRUD data todo dari seorang user. Jalankan perintah artisan berikut. Flag `--api` digunakan untuk menyatakan bahwa resource controller yang dibuat merupakan api.

```
php artisan make:controller Api/TodoController --api --model=Todo
```

5. Buka model Todo dan lengkapi definisi relasi dengan model User. Lengkapi atribut fillable. Informasi kolom user_id tidak dibutuhkan sehingga perlu disembunyikan dari response. Dikarenakan representasi tipe data boolean pada database bernilai angka (0 dan 1), tambahkan casting atribut done dengan boolean. Hasil akhir kode dapat anda lihat sebagai berikut.

```php
class Todo extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',
        'todo',
        'label',
        'done',
    ];

    protected $hidden = [
        'user_id',
    ];

    protected $casts = [
        'done' => 'boolean',
    ];

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

6. Tambahkan fungsi todos pada model User yang menyatakan relasi dengan model Todo. Perhatikan kode berikut.

```php
public function todos()
{
    return $this->hasMany(Todo::class);
}
```

7. Untuk memudahkan proses validasi, buatlah custom request. Jalankan perintah berikut

```
php artisan make:request TodoRequest
```

8. Ubah parent/base class dari TodoRequest menjadi ApiRequest.

```php
class TodoRequest extends ApiRequest
```

9. Lengkapi fungsi authorize sehingga hanya bisa mengijinkan request jika http method menggunakan POST atau perubahan todo dilakukan oleh pemilik todo. Perhatikan kode berikut.

```php
public function authorize()
{
    if ($this->method() == Request::METHOD_POST)
        return true;
    $todo = $this->route('todo');
    return auth()->user()->id == $todo->user_id;
}
```

10. Lengkapi juga fungsi rules dalam TodoRequest menjadi seperti berikut.

```php
public function rules()
{
    return [
        'todo' => 'required|string|max:255',
        'label' => 'nullable|string',
        'done' => 'nullable|boolean',
    ];
}
```

11. Pada proses CRUD pada resource, ada kalanya resource tersebut tidak ditemukan. Untuk memastikan bahwa respon yang dikembalikan dalam format API, modifikasi penanganan exception untuk mengembalikan respon json jika pemanggilan dilakukan pada API. Ubah

file Handler.php pada app/Exceptions/Handler.php dan override fungsi render. Perhatikan kode berikut.

```php
public function render($request, $exception)
{
    if ($exception instanceof ModelNotFoundException && $request->wantsJson()) {
        return response()->json(
            ['message' => 'Not Found!'],
            Response::HTTP_NOT_FOUND
        );
    }

    return parent::render($request, $exception);
}
```

12. Buka file Api\TodoController.php, kemudian tambahkan trait ApiResponse.

```php
class TodoController extends Controller
{

    use ApiResponse;
```

13. Lengkapi proses untuk melakukan CRUD pada resource Todo. Logika untuk melakukan CRUD pada RESTful tidak ada perbedaan. Yang membedakan hanyalah respon berupa json bukan format html lagi. Sehingga konsep-konsep yang sebelumnya telah dipelajari dapat diaplikasikan juga dalam RESTful API. Perhatikan kode berikut untuk implementasi akhir CRUD pada resource Todo.

```php
public function index()
{
    $user = auth()->user();
    $todos = Todo::with('user')
        ->where('user_id', $user->id)
        ->get();

    return $this->apiSuccess($todos);
}
```

```php
public function store(TodoRequest $request)
{
    $request->validated();

    $user = auth()->user();
    $todo = new Todo($request->all());
    $todo->user()->associate($user);
    $todo->save();

    return $this->apiSuccess($todo->load('user'));
}
```

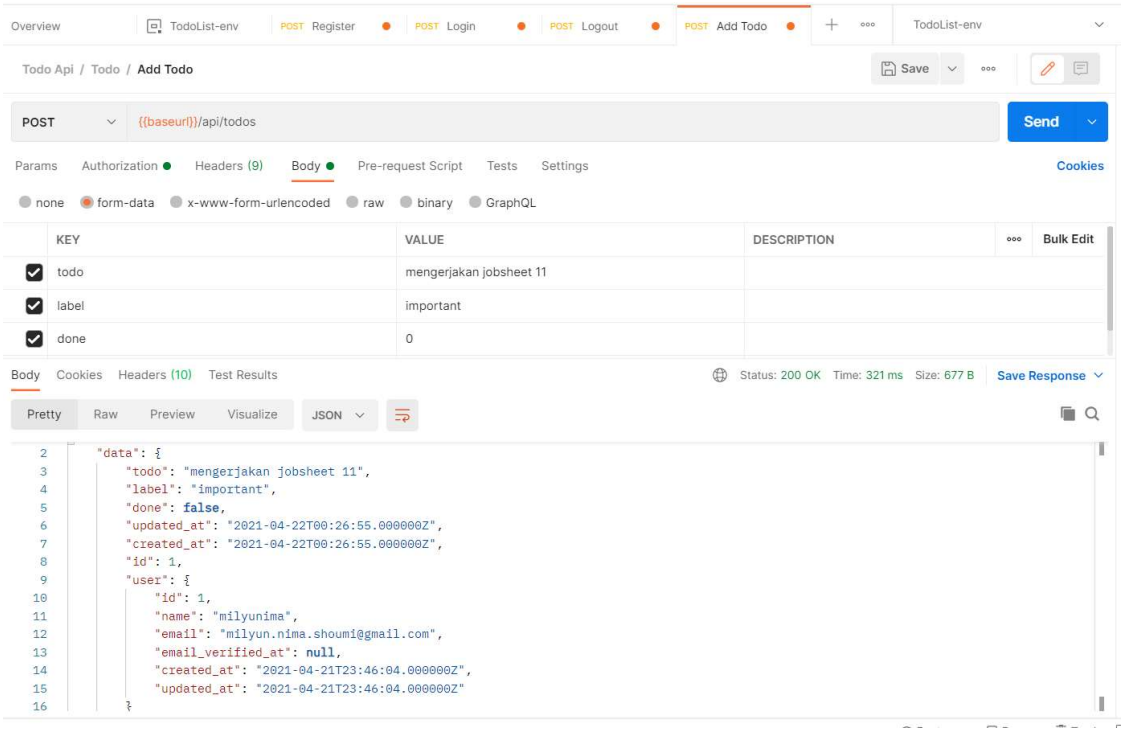
```php
public function show(Todo $todo)
{
    return $this->apiSuccess($todo->load('user'));
}
```

```php
public function update(TodoRequest $request, Todo $todo)
{
    $request->validated();
    $todo->todo = $request->todo;
    $todo->label = $request->label;
    $todo->done = $request->done;
    $todo->save();
    return $this->apiSuccess($todo->load('user'));
}

public function destroy(Todo $todo)
{
    if (auth()->user()->id == $todo->user_id) {
        $todo->delete;
        return $this->apiSuccess($todo);
    }
    return $this->apiError(
        'Unauthorized',
        Response::HTTP_UNAUTHORIZED
    );
}
```

14. Tambahkan route resource pada route/api.php. Gunakan apiResource untuk melakukan penambahan route.

```php
Route::middleware(['auth:sanctum'])->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::apiResource('/todos', TodoController::class);
});
```

15. Buka aplikasi postman dan lakukan percobaan pada CRUD yang telah diselesaikan. Jangan lupa melakukan git commit.



~ Tetap Semangat Mengerjakan ~