

Nama	: Dini Elminingtyas
Kelas/No.absen	: SIB 2B / 07
NIM	: 2341760180

PEMROGRAMAN WEB LANJUT

JOBSHEET 10

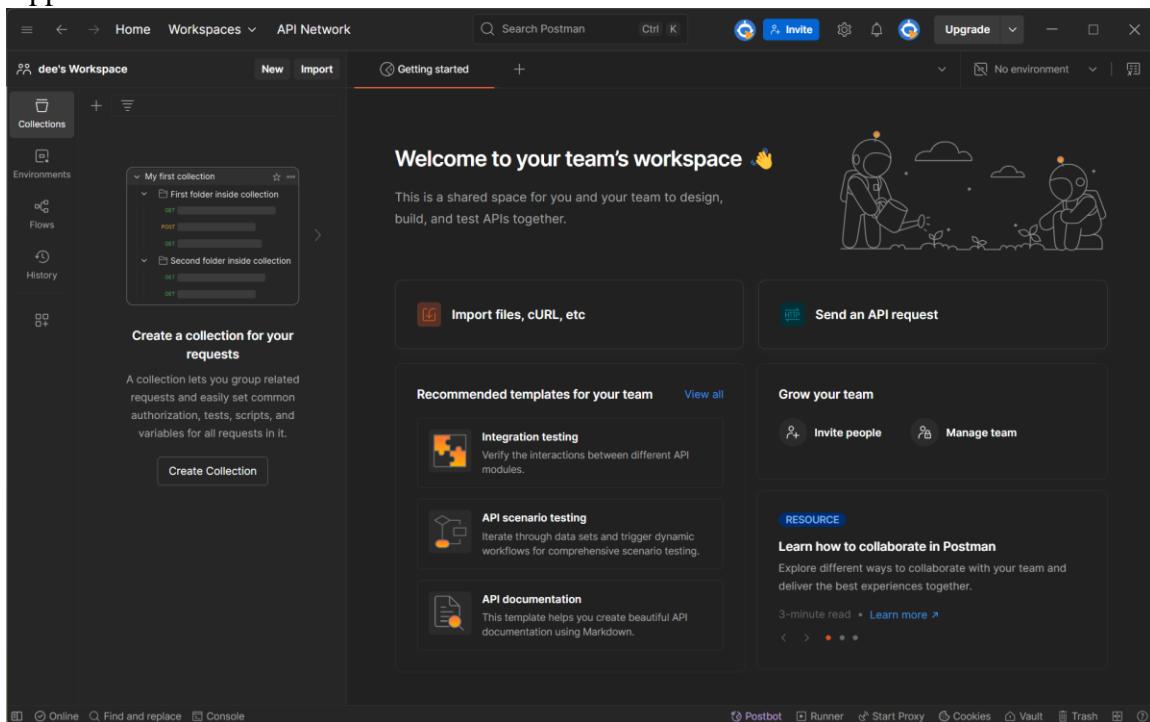
RESTFULL API

Praktikum 1 - Membuat RESTful API Register

- Sebelum memulai membuat REST API, terlebih dahulu download aplikasi Postman di <https://www.postman.com/downloads>.

Aplikasi ini akan digunakan untuk mengerjakan semua tahap praktikum pada Jobsheet ini.

⇒ Apps



- Lakukan instalasi JWT dengan mengetikkan perintah berikut:

```
composer require tymon/jwt-auth:2.1.1
```

Pastikan Anda terkoneksi dengan internet.

⇒ Source code

```
PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> composer require tymon/jwt-auth:2.1.1
./composer.json has been updated
Running composer update tymon/jwt-auth
Loading composer repositories with package information
Updating dependencies
Lock file operations: 4 installs, 0 updates, 0 removals
```

- Setelah berhasil menginstall JWT, lanjutkan dengan publish konfigurasi file dengan perintah berikut:

```
php artisan vendor:publish --  
provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
```

⇒ Source code

```
PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"  
INFO Publishing assets.  
Copying file [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\vendor\tymon\jwt-auth\config\config.php] to [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\config\jwt.php]  
[PWL_POS] DONE
```

- Jika perintah di atas berhasil, maka kita akan mendapatkan 1 file baru yaitu config/jwt.php. Pada file ini dapat dilakukan konfigurasi jika memang diperlukan.

⇒ File



- Setelah itu jalankan perintah berikut untuk membuat secret key JWT.

```
php artisan jwt:secret
```

Jika berhasil, maka pada file .env akan ditambahkan sebuah baris berisi nilai key JWT_SECRET.

⇒ Source code

```
PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan jwt:secret  
jwt-auth secret [fDsIN6RxfbmmpDEjgSAzWATCvxh34QoqCYgdm0ehYZ8m74vhvBrgZ61BsRkQRAlpU] set successfully.
```

- Selanjutnya lakukan konfigurasi guard API. Buka config/auth.php. Ubah bagian 'guards' menjadi seperti berikut.

```
'guards' => [  
    'web' => [  
        'driver' => 'session',  
        'provider' => 'users',  
    ],  
    'api' => [  
        'driver' => 'jwt',  
        'provider' => 'users',  
    ],  
,
```

⇒ Source code

```
38     'guards' => [  
39         'web' => [  
40             'driver' => 'session',  
41             'provider' => 'users',  
42         ],  
43         'api' => [  
44             'driver' => 'jwt',  
45             'provider' => 'users',  
46         ],  
47     ],
```

7. Kita akan menambahkan kode di model UserModel, ubah kode seperti berikut:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Tymon\JWTAuth\Contracts\JWTSubject;
use Illuminate\Foundation\Auth\User as Authenticatable;

class UserModel extends Authenticatable implements JWTSubject
{

    public function getJWTIdentifier(){
        return $this->getKey();
    }

    public function getJWTCustomClaims(){
        return [];
    }

    protected $table = 'm_user';
    protected $primaryKey = 'user_id';
```

⇒ Source code

```
app > Models > UserModel.php > UserModel
  1  <?php
  2
  3  namespace App\Models;
  4
  5  use Illuminate\Database\Eloquent\Model;
  6  use Illuminate\Database\Eloquent\Factories\HasFactory;
  7  use Illuminate\Database\Eloquent\Relations\BelongsTo;
  8  use Tymon\JWTAuth\Contracts\JWTSubject;
  9  use Illuminate\Foundation\Auth\User as Authenticatable; // implementasi class Authenticatable
10
11  class UserModel extends Authenticatable implements JWTSubject
12  {
13      use HasFactory;
14      public function getJWTIdentifier(){
15          return $this->getKey();
16      }
17
18      public function getJWTCustomClaims(){
19          return [];
20      }
21
22      protected $table = 'm_user';
23      protected $primaryKey = 'user_id';
24      protected $fillable = ['username', 'password', 'nama', 'level_id', 'created_at', 'updated_at', 'user_profile_picture'];
25      protected $hidden = ['password']; // jangan di tampilkan saat select
26      protected $casts = ['password' => 'hashed']; // casting password agar otomatis di hash
27  }
```

Hasil yang didapatkan adalah menampilkan halaman form login dikarenakan ketika pada tahapan sebelumnya tidak mengirimkan ke dalam server untuk login. Sehingga pada pengujian hanya menampilkan halaman form login.

8. Berikutnya kita akan membuat controller untuk register dengan menjalankan perintah berikut.

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

Jika berhasil maka akan ada tambahan controller pada folder Api dengan nama RegisterController.

⇒ Source code

```
PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api/RegisterController
[INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api\RegisterController.php] created successfully.
```

9. Buka file tersebut, dan ubah kode menjadi seperti berikut.

```
1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Models\UserModel;
6 use App\Http\Controllers\Controller;
7 use Illuminate\Http\Request;
8 use Illuminate\Support\Facades\Validator;
9
10 class RegisterController extends Controller
11 {
12     public function __invoke(Request $request)
13     {
```

⇒ Source code

```

app > Http > Controllers > Api > RegisterController.php > ...
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Models\UserModel;
6  use App\Http\Controllers\Controller;
7  use Illuminate\Http\Request;
8  use Illuminate\Support\Facades\Validator;
9
10 class RegisterController extends Controller
11 {
12     public function __invoke(Request $request)
13     {
14         //set validation
15         $validator = Validator::make($request->all(), [
16             'username' => 'required',
17             'nama' => 'required',
18             'password' => 'required|min:5|confirmed',
19             'level_id' => 'required'
20         ]);
21
22         //if validations fails
23         if($validator->fails()){
24             return response()->json($validator->errors(), 422);
25         }
26
27         //create user
28         $user = UserModel::create([
29             'username' => $request->username,
30             'nama' => $request->nama,
31             'password' => bcrypt($request->password),
32             'level_id' => $request->level_id,
33         ]);
34
35         //return response JSON user is created
36         if($user){
37             return response()->json([
38                 'success' => true,
39                 'user' => $user,
40             ], 201);
41         }
42
43         //return JSON process insert failed
44         return response()->json([
45             'success' => false,
46         ], 409);
47     }
48 }

```

10. Selanjutnya buka routes/api.php, ubah semua kode menjadi seperti berikut.

```

<?php

use App\Http\Controllers\Api\RegisterController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
*/

Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');

```

⇒ Source code

```
routes > api.php
1  <?php
2
3  | use App\Http\Controllers\Api\RegisterController;
4  | use Illuminate\Http\Request;
5  | use Illuminate\Support\Facades\Route;
6
7  /*
8  |-----
9  | API Routes
10 |-----
11 |
12 | Here is where you can register API routes for your application. These
13 | routes are loaded by the RouteServiceProvider and all of them will
14 | be assigned to the "api" middleware group. Make something great!
15 |
16 */
17
18 | Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
19 |
```

11. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman.

Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/register serta method POST. Klik Send.

The screenshot shows the Postman interface with a failed API request. The request details are as follows:

- Method: POST
- URL: localhost/PWL_POS/public/api/register
- Status: 422 Unprocessable Content
- Time: 272 ms
- Size: 534 B

The 'Body' tab shows a JSON response with validation errors:

```
1
2   "username": [
3     "The username field is required."
4   ],
5   "nama": [
6     "The nama field is required."
7   ],
8   "password": [
9     "The password field is required."
10 ]
11
```

Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:8000/api/register`. The response status is 422 Unprocessable Entity. The error message in the response body is:

```
{
  "username": [
    "The username field is required."
  ],
  "name": [
    "The name field is required."
  ],
  "password": [
    "The password field is required."
  ],
  "level_id": [
    "The level id field is required."
  ]
}
```

12. Sekarang kita coba masukkan data. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data registrasi menggunakan nilai yang Anda inginkan.

The screenshot shows the Postman interface with a POST request to `localhost/PWL_POS/public/api/register`. The **Body** tab is selected and set to **form-data**. The data entered is:

Key	Value
<input checked="" type="checkbox"/> username	penggunasatu
<input checked="" type="checkbox"/> nama	Pengguna 1
<input checked="" type="checkbox"/> password	12345
<input checked="" type="checkbox"/> password_confirmation	12345
<input checked="" type="checkbox"/> level_id	2

The response status is 201 Created. The response body is:

```

1   "success": true,
2   "user": {
3     "username": "penggunasatu",
4     "nama": "Pengguna 1",
5     "password": "$2y$12$Eb2SrV1jsykINytYGtrHi0DVAKcK5p6EgnZnmbChkPicIu7S0QJJu",
6     "level_id": "2",
7     "updated_at": "2024-04-22T15:56:04.000000Z",
8     "created_at": "2024-04-22T15:56:04.000000Z",
9     "user_id": 17
10

```

Setelah klik tombol Send, jika berhasil maka akan keluar pesan sukses seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with a successful API call. The URL is `http://127.0.0.1:8000/api/register`. The response status is `201 Created` with a response time of `1.26 s` and a size of `498 B`. The response body is a JSON object:

```
1 {  
2   "success": true,  
3   "user": {  
4     "username": "penggunasatu",  
5     "nama": "Pengguna 1",  
6     "level_id": "2",  
7     "updated_at": "2025-04-22T04:09:58.000000Z",  
8     "created_at": "2025-04-22T04:09:58.000000Z",  
9     "user_id": 32  
10   }  
11 }
```

13. Lakukan commit perubahan file pada Github.

⇒ commit



Praktikum 2 – Membuat RESTful API Login

1. Kita buat file controller dengan nama LoginController.

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

Jika berhasil maka akan ada tambahan controller pada folder Api dengan nama LoginController.

⇒ create controller

```
PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api/LoginController
[INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api>LoginController.php] created successfully.
```

2. Buka file tersebut, dan ubah kode menjadi seperti berikut.

```
1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Http\Request;
7 use Illuminate\Support\Facades\Validator;
8
```

⇒ controller

```
app > Http > Controllers > Api > LoginController.php > ...
1 <?php
2 namespace App\Http\Controllers\Api;
3
4 use App\Http\Controllers\Controller;
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Validator;
7
8 class LoginController extends Controller
9 {
10     public function __invoke(Request $request)
11     {
12         //set validation
13         $validator = Validator::make($request->all(), [
14             'username' => 'required',
15             'password' => 'required',
16         ]);
17
18         //if validation fails
19         if ($validator->fails()) {
20             return response()->json($validator->errors(), 422);
21         }
22
23         //get credentials from request
24         $credentials = $request->only('username', 'password');
25
26         //if auth failed
27         if (!$token = auth()->guard('api')->attempt($credentials)) {
28             return response()->json([
29                 'success' => false,
30                 'message' => 'Username atau Password Anda salah'
31             ], 401);
32         }
33
34         //if auth success
35         return response()->json([
36             'success' => true,
37             'user' => auth()->guard('api')->user(),
38             'token' => $token
39         ], 200);
40     }
41 }
```

3. Berikutnya tambahkan route baru pada file api.php yaitu /login dan /user.

```
use App\Http\Controllers\Api\LoginController;

Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
Route::post('/login', App\Http\Controllers\Api\LoginController::class)->name('login');
Route::middleware('auth:api')->get('/user', function (Request $request) {
    return $request->user();
});
```

⇒ route

```
routes > api.php > ...
1  <?php
2
3  use App\Http\Controllers\Api\RegisterController;
4  use App\Http\Controllers\Api\LoginController;
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\Route;
7
8  /*
9  | -----
10 | API Routes
11 | -----
12 |
13 | Here is where you can register API routes for your application. These
14 | routes are loaded by the RouteServiceProvider and all of them will
15 | be assigned to the "api" middleware group. Make something great!
16 |
17 */
18
19 Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
20 Route::post('/login', App\Http\Controllers\Api\LoginController::class)->name('login');
21 <? Route::middleware('auth:api')->get('/user', function (Request $request) {
22     return $request->user();
23 });
24 |
```

4. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/login serta method POST. Klik Send.

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** localhost/PWL_POS/public/api/login
- Headers:** (7)
- Body:** (empty)
- Response Status:** 422 Unprocessable Content
- Response Time:** 563 ms
- Response Size:** 495 B
- Response Content:** A JSON object indicating validation errors:


```

1   "username": [
2     "The username field is required."
3   ],
4   "password": [
5     "The password field is required."
6   ]
7 
```

Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with a collection named "dees Workspace". A POST request is being made to "http://127.0.0.1:8000/api/login". The "Body" tab is selected, showing JSON input with two errors: "username" is required and "password" is required. The response status is 422 Unprocessable Content.

- Selanjutnya, isikan username dan password sesuai dengan data user yang ada pada database. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data user. Klik tombol Send, jika berhasil maka akan keluar tampilan seperti berikut. Copy nilai token yang diperoleh pada saat login karena akan diperlukan pada saat logout.

The screenshot shows the Postman interface with a POST request to "localhost/PWL_POS/public/api/login". The "Body" tab is selected with "form-data" chosen. Two fields are filled: "username" with "penggunasatu" and "password" with "12345". The response status is 200 OK, indicating success. The response body is a JSON object containing a token.

```
1 "success": true,
2 "user": {
3     "user_id": 17,
4     "level_id": 2,
5     "username": "penggunasatu",
6     "name": "Pengguna 1",
7     "password": "$2y$12$Eb2SzV1jsykINytYGtzHiODVAKcK5p6EgnZnmbChkPicIu750QJJu",
8     "created_at": "2024-04-22T15:56:04.000000Z",
9     "updated_at": "2024-04-22T15:56:04.000000Z"
10 },
11 "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJpc3MiOiJodHRw018vbG9jYWxob3N0L1BXTF9QT1MtbWFpbihwdWJsaWVXBpL2xvZ2luIiwiaWF0I"
```

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with a successful API call. The URL is `http://127.0.0.1:8000/api/login`. The response status is `200 OK` with a response time of `1.24 s` and a size of `847 B`. The response body is a JSON object:

```
1 {  
2   "success": true,  
3   "user": {  
4     "user_id": 32,  
5     "level_id": 2,  
6     "username": "penggunasatu",  
7     "nama": "Pengguna 1",  
8     "user_profile_picture": null,  
9     "created_at": "2025-04-22T04:09:58.000000Z",  
10    "updated_at": "2025-04-22T04:09:58.000000Z"  
11  },  
12  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRw018vMTI3LjAuMC4xOjgwMDAvYXBpL2xvZ2luIiwk"
```

6. Lakukan percobaan yang untuk data yang salah dan berikan screenshoot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with an unauthorized response. The URL is `http://127.0.0.1:8000/api/login`. The response status is `401 Unauthorized` with a response time of `700 ms` and a size of `381 B`. The response body is a JSON object:

```
1 {  
2   "success": false,  
3   "message": "Username atau Password Anda salah"  
4 }
```

7. Coba kembali melakukan login dengan data yang benar. Sekarang mari kita coba menampilkan data user yang sedang login menggunakan URL `localhost/PWL_POS/public/api/user` dan method GET. Jelaskan hasil dari percobaan tersebut.

⇒ postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (empty), 'Environments' (empty), 'Flows' (empty), and 'History' (empty). The main area shows a list of recent requests under 'Today'. A specific GET request to `http://127.0.0.1:8000/api/user` is selected. The 'Params' tab is active, showing an empty table for query parameters. The 'Body' tab shows the response body as an HTML document. The response details at the bottom indicate a `200 OK` status with a duration of 433 ms and a size of 7.83 KB. The HTML content of the response is as follows:

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="utf-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7      <title>Login Pengguna</title>
8      <!-- Google Font: Source Sans Pro -->
9      <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,400i,700&
10 ack">
11      <!-- Font Awesome -->
12      <link rel="stylesheet" href="http://127.0.0.1:8000/adminlte/plugins/fontawesome-free/css/all.min.css">
13      <!-- iCheck bootstrap -->
14      <link rel="stylesheet" href="http://127.0.0.1:8000/adminlte/plugins/iCheck-bootstrap/iCheck-bootstrap.&
15      <!-- SweetAlert2 -->
16      <link rel="stylesheet"
17          href="http://127.0.0.1:8000/adminlte/plugins/sweetalert2-theme-bootstrap-4/bootstrap-4.min.css">
18      <!-- Theme style -->
19      <link rel="stylesheet" href="http://127.0.0.1:8000/adminlte/dist/css/adminlte.min.css">

```

8. Lakukan commit perubahan file pada Github.

⇒ commit



Praktikum 3 – Membuat RESTful API Logout

1. Tambahkan kode berikut pada file .env

```
JWT_SHOW_BLACKLIST_EXCEPTION=true
```

⇒ .env

```
60  
61     JWT_SECRET=fDsIN6RxfbmpDEjgSAzWATCvxh34QoqCYgdm0ehYZ8m74vhvBrqZ61BsRkQRAJpU  
62     JWT_SHOW_BLACKLIST_EXCEPTION=true  
63
```

2. Buat Controller baru dengan nama LogoutController.

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

```
• PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api/LogoutController  
⇒ [INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api\LogoutController.php] created successfully.
```

3. Buka file tersebut dan ubah kode menjadi seperti berikut.

```
1 <?php  
2  
3 namespace App\Http\Controllers\Api;  
4 use Illuminate\Http\Request;  
5 use App\Http\Controllers\Controller;  
6 use Tymon\JWTAuth\Facades\JWTAuth;  
7 use Tymon\JWTAuth\Exceptions\JWTException;  
8 use Tymon\JWTAuth\Exceptions\TokenExpiredException;  
9 use Tymon\JWTAuth\Exceptions\TokenInvalidException;  
10  
11 class LogoutController extends Controller  
12 {  
13     public function __invoke(Request $request)  
14     {  
15         //remove token  
16         $removeToken = JWTAuth::invalidate(JWTAuth::getToken());  
17  
18         if($removeToken) {  
19             //return response JSON  
20             return response()->json([  
21                 'success' => true,  
22                 'message' => 'Logout Berhasil!',  
23             ]);  
24         }  
25     }  
26 }
```

⇒ source code

```
app > Http > Controllers > Api > LogoutController.php > LogoutController
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use Illuminate\Http\Request;
6  use App\Http\Controllers\Controller;
7  use Tymon\JWTAuth\Facades\JWTAuth;
8  use Tymon\JWTAuth\Exceptions\JWTException;
9  use Tymon\JWTAuth\Exceptions\TokenExpiredException;
10 use Tymon\JWTAuth\Exceptions\TokenInvalidException;
11
12 class LogoutController extends Controller
13 {
14     public function __invoke(Request $request)
15     {
16         //remove token
17         $removeToken = JWTAuth::invalidate(JWTAuth::getToken());
18
19         if($removeToken) {
20             //return response JSON
21             return response()->json([
22                 'success' => true,
23                 'message' => 'Logout Berhasil!',
24             ]);
25         }
26     }
27 }
```

4. Lalu kita tambahkan routes pada api.php

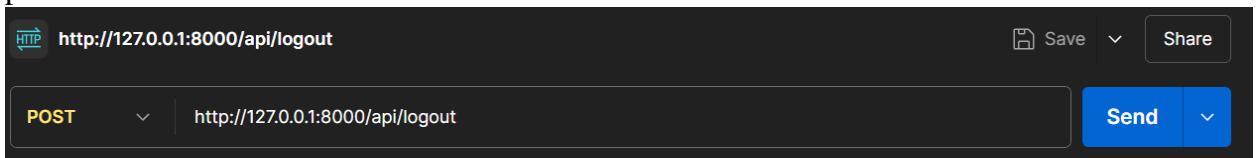
```
Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');
```

⇒ route

```
24 | Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');
25 |
```

5. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/logout serta method POST.

⇒ postman



6. Isi token pada tab Authorization, pilih Type yaitu Bearer Token. Isikan token yang didapat saat login. Jika sudah klik Send.

The screenshot shows the Postman interface for a POST request to `http://localhost/PWL_POS/public/api/logout`. The 'Authorization' tab is active, with 'Type' set to 'Bearer Token'. A red box highlights the token input field containing a long string of characters. The response status is 200 OK, and the response body is:

```

1
2   "success": true,
3   "message": "Logout Berhasil!"
4

```

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface for a POST request to `http://127.0.0.1:8000/api/logout`. The 'Authorization' tab is active, with 'Auth Type' set to 'Bearer Token'. A red box highlights the token input field containing a long string of characters. The response status is 200 OK, and the response body is:

```

1
2   "success": true,
3   "message": "Logout Berhasil!"
4

```

7. Lakukan commit perubahan file pada Github.

⇒ commit

The screenshot shows a GitHub commit history for a repository named 'Minggu 10 (PWL_POS)'. The commit message is 'praktikum 3. api logout'.

Praktikum 4 – Implementasi CRUD dalam RESTful API

1. Pertama, buat controller untuk mengolah API pada data level.

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

⇒ make controller

```
PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api/LevelController
[INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api\LevelController.php] created successfully.
```

2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya.

```
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\LevelModel;

class LevelController extends Controller
{
    public function index()
    {
        return LevelModel::all();
    }
}
```

⇒ controller

```
app > Http > Controllers > Api > LevelController.php > ...
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use App\Models\LevelModel;
7  use Illuminate\Http\Request;
8
9  class LevelController extends Controller
10 <?
11     public function index(){
12         return LevelModel::all();
13     }
14     public function store(Request $request)
15     {
16         $level = LevelModel::create($request->all());
17         return response()->json($level, 201);
18     }
19
20     public function show(LevelModel $level)
21     {
22         return LevelModel::find($level);
23     }
24
25     public function update(Request $request, LevelModel $level)
26     {
27         $level->update($request->all());
28         return LevelModel::find($level);
29     }
30
```

3. Kemudian kita lengkapi routes pada api.php.

```
use App\Http\Controllers\Api\LevelController;

Route::get('levels', [LevelController::class, 'index']);
Route::post('levels', [LevelController::class, 'store']);
Route::get('levels/{level}', [LevelController::class, 'show']);
Route::put('levels/{level}', [LevelController::class, 'update']);
Route::delete('levels/{level}', [LevelController::class, 'destroy']);
```

⇒ routes

```
4  use App\Http\Controllers\Api\LoginController;
5  use App\Http\Controllers\Api\LevelController;
6  use Illuminate\Http\Request;
7  use Illuminate\Support\Facades\Route;
8
9  /*
10  |--------------------------------------------------------------------------
11  | API Routes
12  |--------------------------------------------------------------------------
13  |
14  | Here is where you can register API routes for your application. These
15  | routes are loaded by the RouteServiceProvider and all of them will
16  | be assigned to the "api" middleware group. Make something great!
17  |
18  */
19
20 Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
21 Route::post('/login', App\Http\Controllers\Api\LoginController::class)->name('login');
22 Route::middleware('auth:api')->get('/user', function (Request $request) {
23     return $request->user();
24 });
25 Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');
26
27 Route::get('levels', [LevelController::class, 'index']);
28 Route::post('levels', [LevelController::class, 'store']);
29 Route::get('levels/{level}', [LevelController::class, 'show']);
30 Route::put('levels/{level}', [LevelController::class, 'update']);
31 Route::delete('levels/{level}', [LevelController::class, 'destroy']);
```

4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/levels dan method GET. **Jelaskan dan berikan screenshoot hasil percobaan Anda.**

⇒ postman

The screenshot shows the Postman interface with a collection named "de's Workspace". A GET request is made to `http://127.0.0.1:8000/api/levels`. The response is successful (200 OK) with a response time of 620 ms and a size of 913 B. The response body is a JSON array with three elements, each representing a level with properties: `level_id`, `level_kode`, `level_nama`, `created_at`, and `updated_at`.

```

[{"level_id": 1, "level_kode": "ADMIN", "level_nama": "Administrator", "created_at": null, "updated_at": "2025-03-23T15:13:20.000000Z"}, {"level_id": 2, "level_kode": "MNG", "level_nama": "Manager", "created_at": null, "updated_at": null}, {"level_id": 3, "level_kode": "SPV", "level_nama": "Supervisor", "created_at": "2024-04-22T21:40:32.000000Z", "updated_at": "2024-04-22T21:40:32.000000Z"}]
  
```

Pengujian yang dijalankan menggunakan method indeks dimana sistem mengambil semua data yang tersedia pada tabel level. Hal tersebut terjadi karena menggunakan sintaks all() untuk proses pengambilan datanya.

- Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL_POS-main/public/api/levels dan method POST seperti di bawah ini.

The screenshot shows the Postman interface with a POST request to `localhost/PWL_POS/public/api/levels`. The request body is defined as `form-data` with two fields: `level_kode` (value: SPV) and `level_nama` (value: Supervisor). The response status is 201 Created with a response time of 276 ms and a size of 531 B. The response body is a JSON object with properties: `level_kode`, `level_nama`, `updated_at`, `created_at`, and `level_id`.

```

{
  "level_kode": "SPV",
  "level_nama": "Supervisor",
  "updated_at": "2024-04-22T21:40:32.000000Z",
  "created_at": "2024-04-22T21:40:32.000000Z",
  "level_id": 4
}
  
```

Jelaskan dan berikan screenshoot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:8000/api/levels`. The request body is set to `form-data` and contains two fields: `level_kode` with value `SPV` and `level_nama` with value `Supervisor`. The response is a `201 Created` status with a JSON payload:

```
1 {
2   "level_kode": "SPV",
3   "level_nama": "Supervisor",
4   "updated_at": "2025-04-22T05:40:44.000000Z",
5   "created_at": "2025-04-22T05:40:44.000000Z",
6   "level_id": 9
7 }
```

6. Berikutnya lakukan percobaan menampilkan detail data. **Jelaskan dan berikan screenshoot hasil percobaan Anda.**

⇒ postman

The screenshot shows the Postman interface with a GET request to `http://127.0.0.1:8000/api/levels`. The response is a `200 OK` status with a JSON payload containing a list of levels:

```
13   "created_at": null,
14   "updated_at": null
15 },
16 [
17   {
18     "level_id": 3,
19     "level_kode": "STP",
20     "level_nama": "Staff/Kasir",
21     "created_at": null,
22     "updated_at": null
23   },
24   {
25     "level_id": 4,
26     "level_kode": "CUS",
27     "level_nama": "Customer",
28     "created_at": "2025-03-02T08:21:06.000000Z",
29     "updated_at": "2025-03-02T08:21:06.000000Z"
30   },
31   {
32     "level_id": 8,
33     "level_kode": "KURIR",
34     "level_nama": "kurir",
35     "created_at": "2025-04-02T04:42:01.000000Z",
36     "updated_at": "2025-04-02T05:00:51.000000Z"
37   },
38   {
39     "level_id": 9,
40     "level_kode": "SPR",
41     "level_nama": "Supervisor",
42     "created_at": "2025-04-22T05:40:44.000000Z",
43     "updated_at": "2025-04-22T05:40:44.000000Z"
44 }
```

7. Jika sudah, kita coba untuk melakukan edit data menggunakan localhost/PWL_POS-main/public/api/levels/{id} dan method PUT. Isikan data yang ingin diubah pada tab Param.

PUT localhost/PWL_POS-main/public/api/levels/4?level_kode=SPR

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
level_kode	SPR		
Key	Value	Description	

Body Cookies Headers (11) Test Results Status: 200 OK Time: 266 ms Size: 528 B Save as example

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "level_id": 4,
4     "level_kode": "SPR",
5     "level_nama": "Supervisor",
6     "created_at": "2024-04-22T21:40:32.000000Z",
7     "updated_at": "2024-04-22T21:48:19.000000Z"
8   }
9 ]

```

Jelaskan dan berikan screenshoot hasil percobaan Anda.

⇒ postman

PUT http://127.0.0.1:8000/api/levels/9?level_kode=SPR

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
level_kode	SPR		
Key	Value	Description	

Body Cookies (2) Headers (10) Test Results Status: 200 OK Time: 738 ms Size: 455 B

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "level_id": 9,
4     "level_kode": "SPR",
5     "level_nama": "Supervisor",
6     "created_at": "2025-04-22T05:40:44.000000Z",
7     "updated_at": "2025-04-22T05:48:24.000000Z"
8   }
9 ]

```

8. Terakhir lakukan percobaan hapus data. Jelaskan dan berikan screenshot hasil percobaan Anda.

⇒ postman

The screenshot shows the Postman interface with a collection named 'dee's Workspace'. A DELETE request is made to `http://127.0.0.1:8000/api/levels/8`. The response is a 200 OK status with the following JSON body:

```
{
  "success": true,
  "message": "Data terhapus"
}
```

9. Lakukan commit perubahan file pada Github.

⇒ commit

The screenshot shows a GitHub commit history. A single commit is visible with the following details:

- Commit message: `praktikum 4. api logout`
- Date: `now`

TUGAS

Implementasikan CRUD API pada tabel lainnya yaitu tabel m_user, m_kategori, dan m_barang

⇒ create controller

```
• PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api\UserController
[INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api\UserController.php] created successfully.

• PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api\BarangController
[INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api\BarangController.php] created successfully.

• PS D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)> php artisan make:controller Api\KategoriController
[INFO] Controller [D:\laragon\www\Pemrograman_Web_Lanjut_2025\Minggu 10 (PWL_POS)\app\Http\Controllers\Api\KategoriController.php] created successfully.
```

⇒ controller

- user

```
app > Http > Controllers > Api > UserController.php > ...
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use App\Models\UserModel;
7  use Illuminate\Http\Request;
8
9  class UserController extends Controller
10 {
11     public function index()
12     {
13         return UserModel::all();
14     }
15
16     public function store(Request $request)
17     {
18         $user = UserModel::create($request->all());
19         return response()->json($user, 201);
20     }
21
22     public function show(UserModel $user)
23     {
24         return $user;
25     }
26
27     public function update(Request $request, UserModel $user)
28     {
29         $user->update($request->all());
30         return $user;
31     }
32
33     public function destroy(UserModel $user)
34     {
35         $user->delete();
36
37         return response()->json([
38             'success' => true,
39             'message' => 'Data terhapus',
40         ]);
41     }
42 }
43
```

- barang

```
app > Http > Controllers > Api > BarangController.php > ...
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use App\Models\BarangModel;
7  use Illuminate\Http\Request;
8
9  class BarangController extends Controller
10 {
11     public function index()
12     {
13         return BarangModel::all();
14     }
15
16     public function store(Request $request)
17     {
18         $barang = BarangModel::create($request->all());
19         return response()->json($barang, 201);
20     }
21
22     public function show(BarangModel $barang)
23     {
24         return $barang;
25     }
26
27     public function update(Request $request, BarangModel $barang)
28     {
29         $barang->update($request->all());
30         return $barang;
31     }
32
33     public function destroy(BarangModel $barang)
34     {
35         $barang->delete();
36
37         return response()->json([
38             'success' => true,
39             'message' => 'Data terhapus',
40         ]);
41     }
42 }
43
```

- kategori

```
app > Http > Controllers > Api > KategoriController.php > ...
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use App\Models\KategoriModel;
7  use Illuminate\Http\Request;
8
9  class KategoriController extends Controller
10 {
11     public function index()
12     {
13         return KategoriModel::all();
14     }
15
16     public function store(Request $request)
17     {
18         $kategori = KategoriModel::create($request->all());
19         return response()->json($kategori, 201);
20     }
21
22     public function show(KategoriModel $kategori)
23     {
24         return $kategori;
25     }
26
27     public function update(Request $request, KategoriModel $kategori)
28     {
29         $kategori->update($request->all());
30         return $kategori;
31     }
32
33     public function destroy(KategoriModel $kategori)
34     {
35         $kategori->delete();
36
37         return response()->json([
38             'success' => true,
39             'message' => 'Data terhapus',
40         ]);
41     }
42 }
43
```

⇒ postman

- user
- post

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, Environments, Flows, APIs, Mock servers, Monitors, Specs, and History. The History section is currently selected. In the main workspace, a collection named "deel's Workspace" is open. A POST request is being made to the URL `http://127.0.0.1:8000/api/users`. The "Body" tab is selected, showing the following form-data:

Key	Type	Value
level_id	Text	2
username	Text	penggunadua
nama	Text	pengguna dua
password	Text	123456

The response at the bottom of the screen shows a 201 Created status with a JSON response body:

```
1  {
2      "level_id": "2",
3      "username": "penggunadua",
4      "nama": "pengguna dua",
5      "updated_at": "2025-04-22T12:01:07.000000Z",
6      "created_at": "2025-04-22T12:01:07.000000Z",
7      "user_id": 33
8  }
```

Get

The screenshot shows the Postman interface with a collection named "dee's Workspace". A GET request is made to `http://127.0.0.1:8000/api/users`. The response body is a JSON array containing two user objects:

```
[{"user_id": 1, "level_id": 1, "username": "admin", "name": "Administrator", "user_profile_picture": "profiles/profile_1_1744477551.jpg", "created_at": null, "updated_at": "2025-04-12T17:05:51.000000Z"}, {"user_id": 2, "level_id": 3, "username": "manager", "name": "Manager", "user_profile_picture": null, "created_at": null, "updated_at": null}]
```

Get/id

The screenshot shows the Postman interface with a collection named "dee's Workspace". A GET request is made to `http://127.0.0.1:8000/api/users/33`. The response body is a JSON object representing a single user:

```
{"user_id": 33, "level_id": 2, "username": "pengguna dua", "name": "pengguna dua", "user_profile_picture": null, "created_at": "2025-04-22T12:01:07.000000Z", "updated_at": "2025-04-22T12:01:07.000000Z"}
```

- barang

post

POST http://127.0.0.1:8000/api/barangs

Key	Value	Description
barang_id	11	
barang_kode	BRG11	
barang_nama	vest	
harga_beli	100000	
harga_jual	150000	
kategori_id	2	

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2     "barang_kode": "BRG11",  
3     "barang_nama": "vest",  
4     "harga_beli": "100000",  
5     "harga_jual": "150000",  
6     "kategori_id": "2",  
7     "updated_at": "2025-04-22T12:12:25.000000Z",  
8     "created_at": "2025-04-22T12:12:25.000000Z",  
9     "barang_id": 14  
10 }
```

Get

GET http://127.0.0.1:8000/api/barangs

Key	Value	Description
Key	Value	Description

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
88     "kategori_id": 1,  
89     "created_at": null,  
90     "updated_at": null  
91 },  
92 [  
93     {"barang_id": 10,  
94     "barang_kode": "BRG010",  
95     "barang_nama": "Mouse",  
96     "harga_beli": 100000,  
97     "harga_jual": 150000,  
98     "kategori_id": 1,  
99     "created_at": null,  
100    "updated_at": null  
101 },  
102     {"barang_id": 14,  
103     "barang_kode": "BRG11",  
104     "barang_nama": "vest",  
105     "harga_beli": 100000,  
106     "harga_jual": 150000,  
107     "kategori_id": 2,  
108     "created_at": "2025-04-22T12:12:25.000000Z",  
109     "updated_at": "2025-04-22T12:12:25.000000Z"  
110 ]
```

Get/id

HTTP Request: GET http://127.0.0.1:8000/api/barangs/14

Body:

```
1 {
2     "barang_id": 14,
3     "barang_kode": "BRG11",
4     "barang_nama": "vest",
5     "harga_beli": 100000,
6     "harga_jual": 150000,
7     "kategori_id": 2,
8     "created_at": "2025-04-22T12:12:25.000000Z",
9     "updated_at": "2025-04-22T12:12:25.000000Z"
10 }
```

- kategori post

HTTP Request: POST http://127.0.0.1:8000/api/kategoris

Body:

```
1 {
2     "kategori_kode": "TOYS",
3     "kategori_nama": "Mainan",
4     "updated_at": "2025-04-22T12:14:54.000000Z",
5     "created_at": "2025-04-22T12:14:54.000000Z",
6     "kategori_id": 15
7 }
```

Get

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar has sections for Collections, Environments, Flows, APIs, Mock servers, Monitors, Specs, and History. Under History, there is a list of recent API calls. The main area shows a GET request to `http://127.0.0.1:8000/api/kategoris`. The Params tab shows a single query parameter `Key` with value `Value`. The Body tab displays the JSON response:

```
44  [
45    {
46      "kategori_id": 8,
47      "kategori_kode": "MNR",
48      "kategori_nama": "Minuman Ringan",
49      "created_at": null,
50      "updated_at": null
51    },
52    {
53      "kategori_id": 12,
54      "kategori_kode": "SPTU",
55      "kategori_nama": "Sepatu",
56      "created_at": "2025-03-13T09:02:21.000000Z",
57      "updated_at": "2025-03-13T09:02:21.000000Z"
58    },
59    {
60      "kategori_id": 15,
61      "kategori_kode": "TOYS",
62      "kategori_nama": "Mainan",
63      "created_at": "2025-04-22T12:14:54.000000Z",
64      "updated_at": "2025-04-22T12:14:54.000000Z"
65    }
]
```

The status bar at the bottom indicates `200 OK`, `188 ms`, and `1.35 KB`.

Get/id

The screenshot shows the Postman interface with a dark theme. The sidebar and history list are identical to the previous screenshot. The main area shows a GET request to `http://127.0.0.1:8000/api/kategoris/15`. The Params tab shows a single query parameter `Key` with value `Value`. The Body tab displays the JSON response:

```
1  {
2    "kategori_id": 15,
3    "kategori_kode": "TOYS",
4    "kategori_nama": "Mainan",
5    "created_at": "2025-04-22T12:14:54.000000Z",
6    "updated_at": "2025-04-22T12:14:54.000000Z"
7  }
```

The status bar at the bottom indicates `200 OK`, `225 ms`, and `460 B`.

