

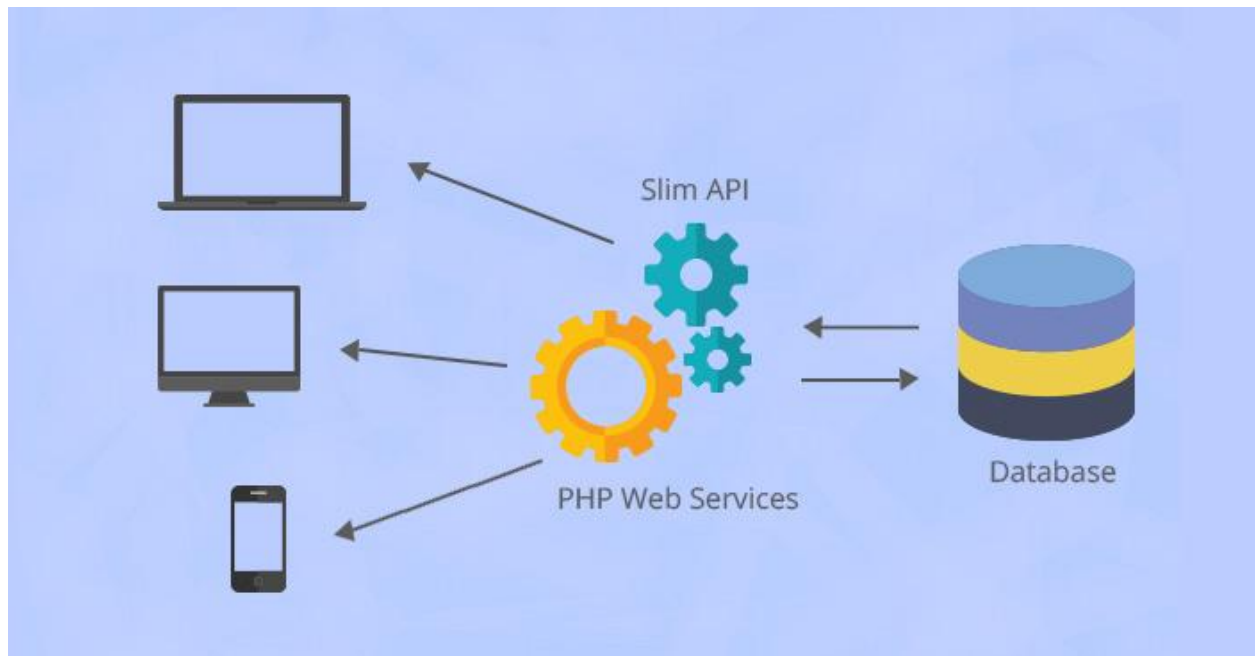
Modul RESTfull API dengan Slim Framework PHP

RestAPI

REST merupakan singkatan dari Representational State Transfer. yang mana sebagai jembatan atau media antara data resource(sumber data/database) dan application interface, entah itu pada mobile device atau desktop. REST menyediakan blok HTTP methods yang dapat digunakan untuk mengubah data. ketika kita bicara mengenai REST API, pada intinya kita akan membuat routes yang akan berputar dengan Object Request dan Response. REST bekerja pada action dan resource. setiap kali action pemanggilan pada URL, maka menjalankan individual method (atau satu set metode) di url tsb.

Slim Framework PHP

Slim merupakan salah satu php micro framework yang dapat membantu untuk pengembangan aplikasi web dan APIs dengan lebih cepat. Salah satu penggunaan yang penting adalah dalam pengembangan REST API. Slim mendukung semua metode HTTP (GET, POST, PUT, DELETE). Slim berisi struktur URL sangat berguna dengan router, middlewares, bodyparser dengan halaman template, cookie yang dienkripsi dan banyak lagi.



Gambar 1. Logic tier dengan Slim Framework

Contoh konsep resource api :

> api/customer/1

> api/customer/add

> api/segment/add

Prepared Tools and Setting dev Environment

Untuk memulai implemetasi API dengan Framework PHP Slim, sebelumnya kita perlu mempersiapkan dulu tools yang akan kita gunakan yakni,

1. XAMPP
2. Sublime/Notepad++(atau bisa menggunakan text editor/IDE yang lain)
3. Composer dependency Management
4. Slim Frameworks.
5. Postman

#1 Initial Step

Kita mulai langkah pertama dengan mempersiapkan dan melakukan konfigurasi tools yang akan mendukung dalam lingkungan pengembangan. Diawali dengan instalasi XAMPP yang akan kita gunakan sebagai peladen, jika belum terinstall. Yang kemudian akan lebih didetailkan pada langkah-langkah berikut ini :

- 1) *install* xampp (kita anggap saja sudah terinstall).
- 2) *installasi* composer (getcomposer.com) > via .exe (OS windows) atau via curl (OS Linux)

!!Sekilas Tentang Composer

Composer merupakan dependency management untuk PHP. waduh, apa lagi itu dependency management mas/pak/buk? oke, kita singgung sedikit dlu ya? jadi,

Dependency Management merupakan sebuah tools yang memecahkan masalah, pada contoh kasus berikut ini :

1. misalnya, Anda sedang ada project yang dimana butuh library. contohnya, library untuk ekspor data ke Excel yaitu PHPExcel dan DomPDF buat ekspor data ke PDF.
2. Library-library tersebut juga butuh library lainnya. Dalam kasus ini, PHPExcel butuh library ext-xml dan DomPDF butuh phenx/php-font-lib. Kasus seperti ini, namanya dependensi. Jadi, bisa disebut PHPExcel memiliki dependensi ext-xml.
3. Anda males download (install) semua library tersebut berikut dependensinya secara manual.



Gambar 2. Logo Composer

Secara garis besar, **cara kerja composer** yakni seperti ini :

1. Anda menulis library apa saja yang dibutuhkan project.
2. Composer mencari versi library yang sesuai dan menginstallnya (download) secara otomatis.

Dengan composer, kita tidak usah repot-repot download library PHP manual satu-persatu. Begitupun dengan proses update library, tidak usah diupdate satu-persatu, cukup ubah satu file (composer.json), jalankan perintah composer update untuk mengupdate semua library.

3) Langkah ketiga, Install Slim Framework dari composer. Ada beberapa cara untuk menginstall Slim dengan composer. Cara yang pertama :

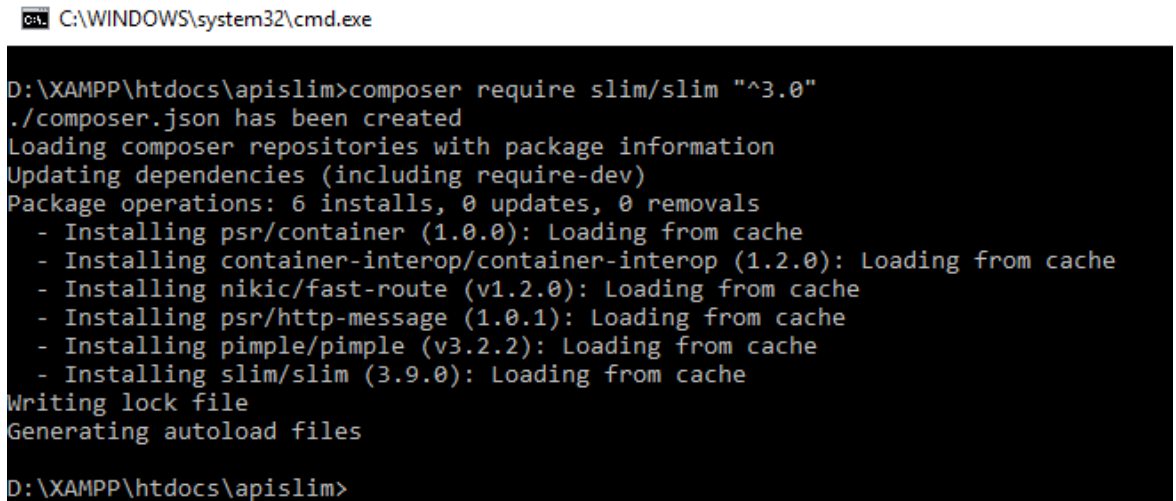
1. Buka command line / terminal, lalu arahkan pada direktori kerja. Misalnya :

```
cd ~\XAMPP\htdocs\apisample
```

2. ketikkan baris kode pada command line (windows) / terminal (linux)

```
composer require slim/slim "^3.0"
```

3. kemudian, tekan Enter untuk melakukan pengunduhan Slim dependency.



```
C:\WINDOWS\system32\cmd.exe

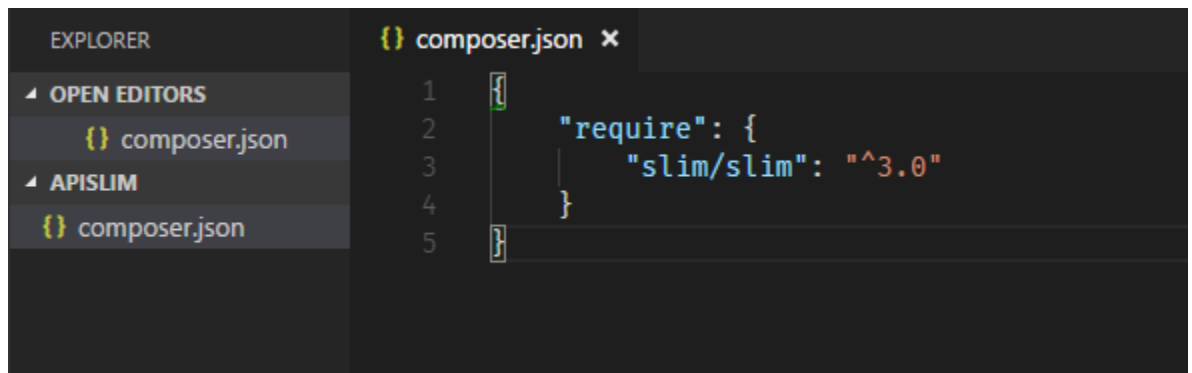
D:\XAMPP\htdocs\apislim>composer require slim/slim "^3.0"
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 6 installs, 0 updates, 0 removals
  - Installing psr/container (1.0.0): Loading from cache
  - Installing container-interop/container-interop (1.2.0): Loading from cache
  - Installing nikic/fast-route (v1.2.0): Loading from cache
  - Installing psr/http-message (1.0.1): Loading from cache
  - Installing pimple/pimple (v3.2.2): Loading from cache
  - Installing slim/slim (3.9.0): Loading from cache
Writing lock file
Generating autoload files
D:\XAMPP\htdocs\apislim>
```

Gambar 3. install composersatu

atau, dengan cara kedua :

1. buka text editor yang akan digunakan, kemudian buat file composer.json yang ada pada direktori kerja (File : ~XAMPP\htdocs\apisample)

2. ketikan baris kode berikut ini pada file composer.json



```
EXPLORER
└─ OPEN EDITORS
    └─ {} composer.json
    └─ APISLIM
        └─ {} composer.json

{} composer.json x
1  {
2      "require": {
3          "slim/slim": "^3.0"
4      }
5  }
```

Gambar 4. Baris kode composer.json

Dengan composer, Anda cukup menyimpan nama dari package yang dibutuhkan dan versinya pada bagian require.

3. Selanjutnya untuk mendownload package tersebut, buka terminal dan arahkan pada lokasi file 'composer.json' pada direktori kerja, lalu jalankan perintah :

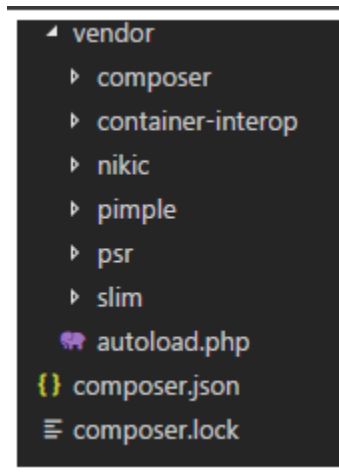
```
composer install
```

4. kemudian, tekan Enter untuk melakukan pengunduhan Slim dependency.

```
D:\XAMPP\htdocs\apislim>composer install
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 6 installs, 0 updates, 0 removals
 - Installing psr/container (1.0.0): Loading from cache
 - Installing container-interop/container-interop (1.2.0): Loading from cache
 - Installing nikic/fast-route (v1.2.0): Loading from cache
 - Installing psr/http-message (1.0.1): Loading from cache
 - Installing pimple/pimple (v3.2.2): Loading from cache
 - Installing slim/slim (3.9.0): Loading from cache
Writing lock file
Generating autoload files
D:\XAMPP\htdocs\apislim>
```

Gambar 5. Install composer dua

Kedua cara diatas hasilnya adalah :



Gambar 6. Hasil instalasi slim via composer

Agar lebih mudah untuk dipahami , berikut penjelasan file dan folder diatas:

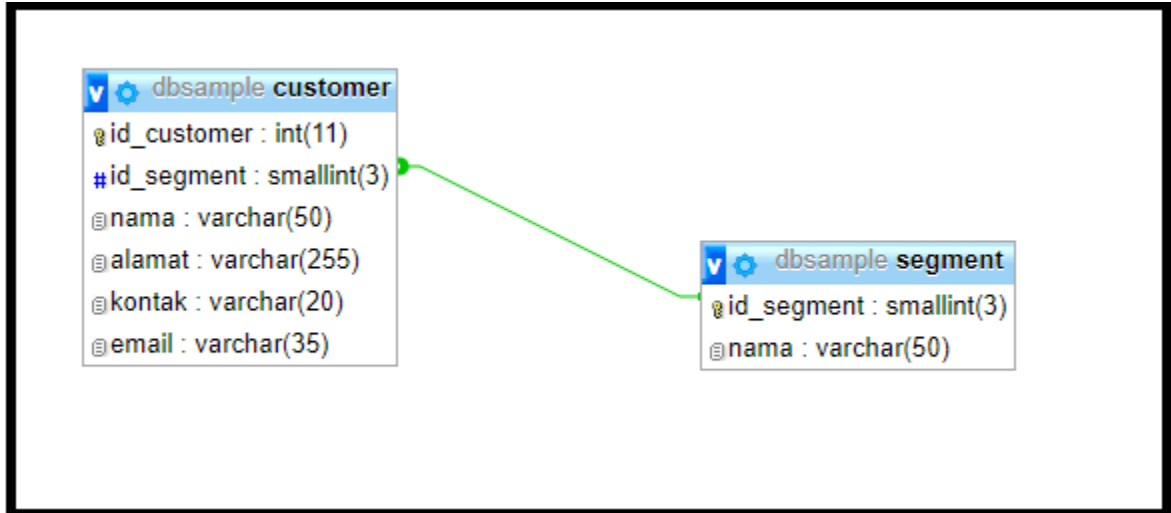
- **composer.json**, ini file yang kita buat, berisi dependensi library dari project kita.
- **composer.lock**, file ini mencatat versi package yang saat ini terinstal.

- **vendor**, folder ini berisi package yang telah kita install. Setiap package yang kita tulis di bagian require akan di download ke folder ini.
- **vendor/autoload.php**, berfungsi memanggil autoloader dari composer.

Salah satu keunggulan dari composer adalah autoloader. Fitur ini berfungsi untuk memanggil class yang sesuai ketika kita membutuhkan class dari suatu library. Jadi, jika banyak library yang kita install kita hanya cukup menulis require untuk file vendor/autoload.php. Dan, autoload ini cukup cerdas dengan hanya me-load class yang kita butuhkan.

#2 The Second Step

Langkah awal pada bagian kedua ini adalah menyiapkan basisdata sederhana berbasis SQL. Banyak basisdata yang bisa digunakan seperti Ms.SQL Server, PostgreSQL, MySQL/MariaDb, ataupun SQLite. Namun kali ini, basisdata yang akan digunakan adalah MySQL. Terdapat 2 tabel yang disiapkan dengan struktur seperti dibawah ini:



Gambar 7. Query table segment

1. Membuat tabel segmet dengan query berikut ini,

```
1 CREATE TABLE segment (  
2     id_segment SMALLINT(3) NOT NULL PRIMARY KEY AUTO_INCREMENT,  
3     nama VARCHAR(50) NOT NULL  
4 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Gambar 8. Query table segment

2. Selanjutnya kita juga buat view untuk table segment :

```
1 CREATE VIEW vsegment AS  
2 SELECT * FROM segment
```

Gambar 9. Query view segmen

3. Membuat tabel customer dengan query berikut ini,

```
1 CREATE TABLE customer (  
2     id_customer INT(11) NOT NULL PRIMARY key AUTO_INCREMENT,  
3     id_segment SMALLINT(3) NOT NULL,  
4     index (id_segment),  
5     FOREIGN key(id_segment) REFERENCES segment(id_segment) on UPDATE  
6     CASCADE on DELETE RESTRICT,  
7     nama VARCHAR(50) NOT NULL,  
8     alamat VARCHAR(255) NOT NULL,  
9     kontak VARCHAR(20) NOT NULL,  
10    email VARCHAR(35) NOT NULL  
11 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Gambar 10. Query table customer

4. Selanjutnya kita juga buat view untuk table customer :

```
1 CREATE VIEW vcustomer AS
2 SELECT customer.id_customer,customer.nama AS customer_name,
   segment.nama AS segment_name, customer.alamat, customer.kontak,
   customer.email FROM customer JOIN segment ON
   customer.id_segment=segment.id_segment
```

Gambar 11. Query view customer

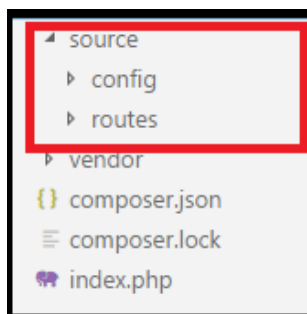
5. Jika basisdata dan table sudah berhasil dibuat, maka kita butuh data dummy untuk mempermudah ketika testing nantinya, langkah selanjutnya ketikkan baris kode berikut ini :

```
1 INSERT INTO segment VALUES (1,'General
   Trait'),(2,'Modern Trait'),(3,'Standart
   Trait');|
```

Gambar 12. Query view customer

#3 The Third Step

Setelah kita selesai dengan basis data dan table, maka selanjutnya kembali pada direktori kerja yang mana pada bagian *intial step* sudah berhasil mengunduh dependensi Slim pada direktori tersebut. Kemudian untuk melakukan pengujian awal dengan Slim Framework, siapkan folder source yang didalamnya terdapat dua folder yakni **folder config** (untuk menyimpan file koneksi database) dan **folder routes** (menyimpan file route) buatlah file **index.php** serta tambahkan baris kode berikut.



Gambar 12. Struktur folder

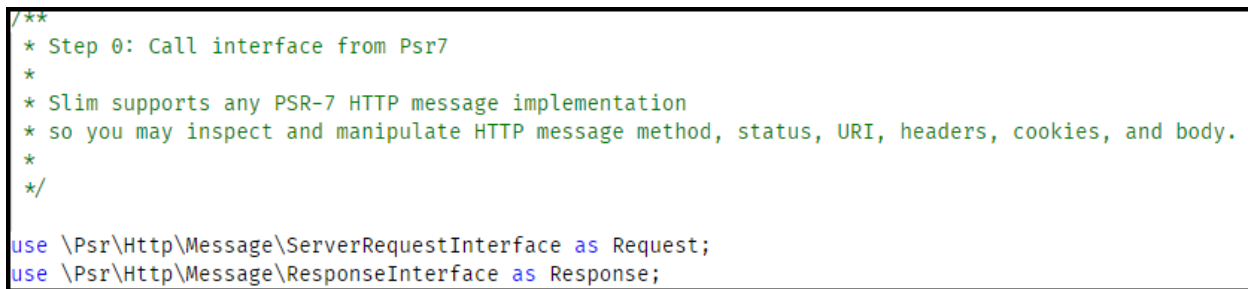


```
index.php
1  <?php
2  use \Psr\Http\Message\ServerRequestInterface as Request;
3  use \Psr\Http\Message\ResponseInterface as Response;
4
5  require 'vendor/autoload.php';
6
7  $app = new \Slim\App;
8  $app->get('/hello/{name}', function (Request $request, Response $response) {
9      $name = $request->getAttribute('name');
10     $response->getBody()->write("Hello, $name");
11
12     return $response;
13 });
14 $app->run();
```

The image shows a code editor window with the file 'index.php'. The code is as follows: Line 1: <?php; Line 2: use \Psr\Http\Message\ServerRequestInterface as Request; Line 3: use \Psr\Http\Message\ResponseInterface as Response; Line 4: (empty); Line 5: require 'vendor/autoload.php'; Line 6: (empty); Line 7: \$app = new \Slim\App; Line 8: \$app->get('/hello/{name}', function (Request \$request, Response \$response) { Line 9: \$name = \$request->getAttribute('name'); Line 10: \$response->getBody()->write("Hello, \$name"); Line 11: Line 12: return \$response; Line 13: }); Line 14: \$app->run(); Annotations: A red '0' with a slash is next to line 3. A red '1' with a slash is next to line 5. A red '2' with a slash is next to line 7. A red '3' with a bracket is next to the function block on lines 8-12.

Gambar 12. Baris kode index.php

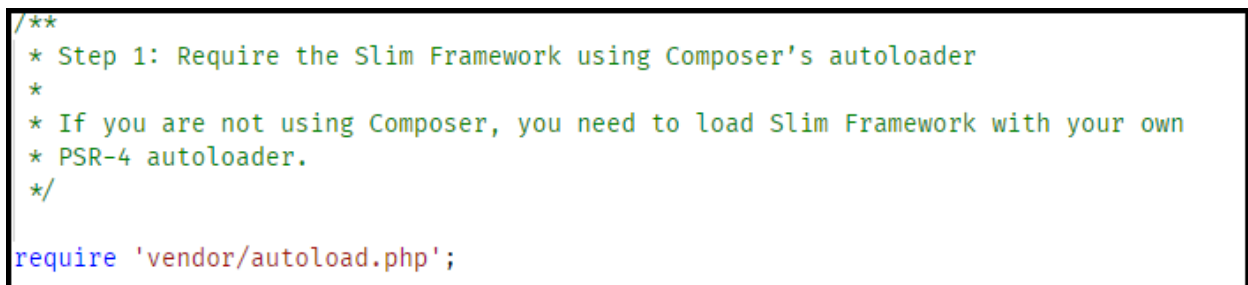
Sedikit penjelasan baris kode diatas dalam komentar berikut:



```
/**
 * Step 0: Call interface from Psr7
 *
 * Slim supports any PSR-7 HTTP message implementation
 * so you may inspect and manipulate HTTP message method, status, URI, headers, cookies, and body.
 */
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;
```

The image shows a code editor window with the following code: Line 1: /** Line 2: * Step 0: Call interface from Psr7 Line 3: * Line 4: * Slim supports any PSR-7 HTTP message implementation Line 5: * so you may inspect and manipulate HTTP message method, status, URI, headers, cookies, and body. Line 6: */ Line 7: use \Psr\Http\Message\ServerRequestInterface as Request; Line 8: use \Psr\Http\Message\ResponseInterface as Response;

Gambar 13. Komentar Baris kode Step 0



```
/**
 * Step 1: Require the Slim Framework using Composer's autoloader
 *
 * If you are not using Composer, you need to load Slim Framework with your own
 * PSR-4 autoloader.
 */
require 'vendor/autoload.php';
```

The image shows a code editor window with the following code: Line 1: /** Line 2: * Step 1: Require the Slim Framework using Composer's autoloader Line 3: * Line 4: * If you are not using Composer, you need to load Slim Framework with your own Line 5: * PSR-4 autoloader. Line 6: */ Line 7: require 'vendor/autoload.php';

Gambar 14. Komentar Baris kode Step 1

```
// instantiate the Slim app object
/**
 * Step 2: Instantiate a Slim application
 *
 * This example instantiates a Slim application using
 * its default settings. However, you will usually configure
 * your Slim application now by passing an associative array
 * of setting names and values into the application constructor.
 */
$app = new \Slim\App;
```

Gambar 15. Komentar Baris kode Step 2

```
/**
 * Step 3: Define the Slim application routes
 *
 * Here we define several Slim application routes that respond
 * to appropriate HTTP request methods. In this example, the second
 * argument for `Slim::get`, `Slim::post`, `Slim::put`, and `Slim::delete`
 * is an anonymous function.
 */
/** implement rest of the methods here */

$app->get('/hello/{name}', function (Request $request, Response $response) {
    $name = $request->getAttribute('name');
    $response->getBody()->write("Hello, $name");

    return $response;
});

/** rest of the methods end here */
```

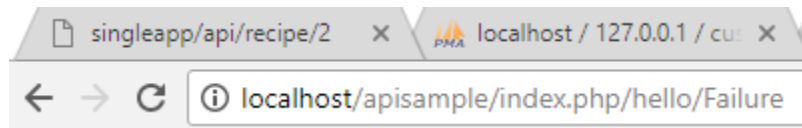
Gambar 16. Komentar Baris kode Step 3

```
/**
 * Step 4: Run the Slim application
 *
 * This method should be called last. This executes the Slim application
 * and returns the HTTP response to the HTTP client.
 */

$app->run();|
```

Gambar 17. Komentar Baris kode Step 4

Langkah berikutnya adalah menguji web yang telah dibuat. Untuk proses read, langsung saja dengan membuka link sesuai dengan route yang dibuat pada method GET



Gambar 18. Pengujian url

Maka hasil yang akan ditampilkan ialah :



Pada bagian route, variable nama (\$nama) yang menyimpan nilai dalam attribute ‘nama’ yang kita berikan pada url, akan ditampilkan atau direspon pada body dan mencetak kalimat “Hello, \$name”

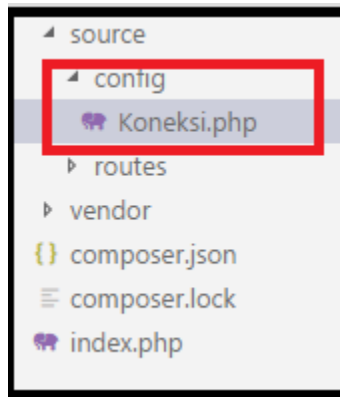
HTTP Rest Service

kita akan mulai membuat rest end points , berikut kita simak GET,POST,PUT, and DELETE type request berikut ini :

#	Route	Method	Type	FullRoute	Description
1	/segment	GET	JSON	http://test.com/api/v1/segment	Get all segment data
2	/segment/{id}	GET	JSON	http://test.com/api/v1/segment/1	Get a single segment data
3	/segment/add	POST	JSON	http://test.com/api/v1/segment/add	Create new record in database
4	/segment/edit/{id}	PUT	JSON	http://test.com/api/v1/segment/edit/2	Update an segment record
5	/segment/delete/{id}	DELETE	JSON	http://test.com/api/v1/delete/1	Delete an segment record

Tabel 1. Web service segment (end points)

Langkah pertama adalah menyiapkan file **Koneksi.php** pada **folder resource > config**. File ini akan digunakan untuk menyambungkan antara basisdata dengan situsweb.



Gambar 19. menyiapkan file koneksi.php

```
1 <?php
2
3 Class Koneksi {
4
5     //set an properties
6     private $dbhost ="localhost";
7     private $dbname ="dbsample";
8     private $username ="root";
9     private $password ="";
10
11     //set Method Connection
12     public function Connection(){
13         //make $mysql_connect_str to store dbhost and dbname
14         $mysql_connect_str = "mysql:host=".$this->dbhost.";dbname=".$this->dbname;
15         //make connection to database
16         $dbConnection = new PDO($mysql_connect_str,$this->username,$this->password);
17         //set error mode
18         $dbConnection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
19
20         return $dbConnection;
21     }
22 }
```

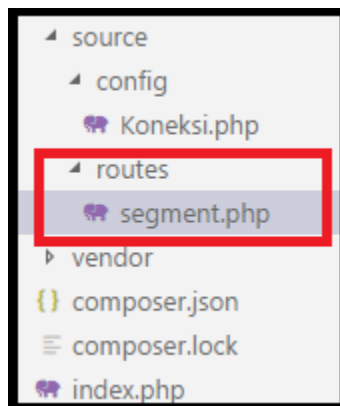
Gambar 20. Baris Kode Class Koneksi

Berikut penjelasan skrip koneksi dengan PDO diatas :

- `$dbConnection = new PDO($mysql_connect_str,$this->username,$this->password);`, membuat objek `$dbConnection` sebagai new PDO yang menjalankan fungsi untuk mengakases host, database name, username, dan password.

- `$dbConnection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);`, mengaktifkan mode error untuk menampilkan exception jika terjadi kesalahan.

Langkah kedua, membuat file **segment.php** didalam folder routes, file ini digunakan sebagai penunjuk arah atau route yang sesuai dengan end point segment. Sementara kita buat route untuk menampilkan data dummy yang ada pada table segment.



Gambar 21. menyiapkan file segment.php

```

<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

//Route select all data
$app->get('/api/v1/segment', function(Request $request, Response $response){

    //set the sql query
    $query = "SELECT * FROM vsegment ORDER BY id_segment asc";

    //make try-catch for error handling
    try {
        //initiate object from Class Koneksi
        $dbase = new Koneksi();
        //call Method Connection in $conn
        $conn = $dbase->Connection();
        //run the sql query
        $state = $conn->query($query);
        //Returns an array containing all of the result set rows
        $segments = $state->fetchAll(PDO::FETCH_OBJ);
        //set connection to null
        $conn = null;
        //show data from $segment to JSON format
        echo json_encode($segments);
    } catch(PDOException $shit){
        //show error messages
        echo '{"error":"text":{"' . $shit->getMessage() . '}}';
    }
});

```

Gambar 22. Baris Kode route getAll data segment

Berikut penjelasan potongan script diatas :

- `$app->get('/api/v1/segment', function(Request $request, Response $response){`, `$app->get()` memanggil route method GET kemudian diberikan route ke 'api/v1/segment' dan callback yang dilewatkan dengan dua argument, yang pertama **Request \$Request** adalah `Psr\Http\Message\ServerRequestInterface` objek yang merepresentasikan request Http saat ini. Lalu yang kedua, **Response \$Response** adalah `Psr\Http\Message\ResponseInterface` objek yang merepresentasikan response Http saat ini.
- `$query = "SELECT * FROM vsegment ORDER BY id_segment asc";`, menyiapkan pernyataan sql untuk menampilkan semua record data yang ada pada view segment (vsegment), kemudian diurutkan berdasarkan id_segment secara Ascending.
- `$dbase = new Koneksi();`, membuat objek \$dbase agar dapat mengakses kelas Koneksi.

- \$conn = \$dbase->Connection(); , mengaktifkan koneksi ke database dengan pemanggilan method Connection() yang ada pada kelas Koneksi.
- \$state = \$conn->query(\$query); , mengeksekusi perintah query didalam variabel \$query.
- \$segments = \$state->fetchAll(PDO::FETCH_OBJ); , mengambil semua data yang terpilih dari dalam view segment dengan fetchAll kemudian menyimpannya dalam variable '\$segments' sebagai array untuk ditampilkan, PDO::FETCH_OBJ, pengembalian sebuah objek anonim dengan nama properti yang sesuai dengan nama-nama kolom yang dikembalikan dalam result set.
- \$conn = null; , menutup koneksi ke database.
- echo json_encode(\$segments); , menampilkan data yang ada didalam variable \$segments yang diubah ke dalam format json.
- catch(PDOException \$shit), menampilkan kesalahan exception diikuti detail kesalahan yang terjadi pada baris berikutnya

```
//select single data segment by id
$app->get('/api/v1/segment/{id}', function(Request $request, Response $response){

    //get value has passed in {id}
    $id = $request->getAttribute('new');

    //set the sql query
    $query = "SELECT * FROM vsegment WHERE id_segment= $id";

    //make try-catch for error handling
    try {
        //initiate object from Class Koneksi
        $dbase = new Koneksi();

        //call Method Connection in $conn
        $conn = $dbase->Connection();

        //run the sql query
        $state = $conn->query($query);

        //Returns an array containing all of the result set rows
        //returns an anonymous object with property names that correspond to the column names returned in
        $segments = $state->fetchAll(PDO::FETCH_OBJ);

        //set connection to null
        $conn = null;

        //show data from $segments to JSON format
        echo json_encode($segments);
    } catch(PDOException $e){
        //show error messages
        echo '{"error":"text":{"'.$e->getMessage().'"}';
    }

});
```

Gambar 23. Baris Kode route getSingle data segment

Langkah ketiga, ubah kode pada index.php dengan menambahkan require 'source/config/Koneksi.php' untuk memanggil file Koneksi.php pada file index.php

```
require 'vendor/autoload.php';  
require 'source/config/Koneksi.php';
```

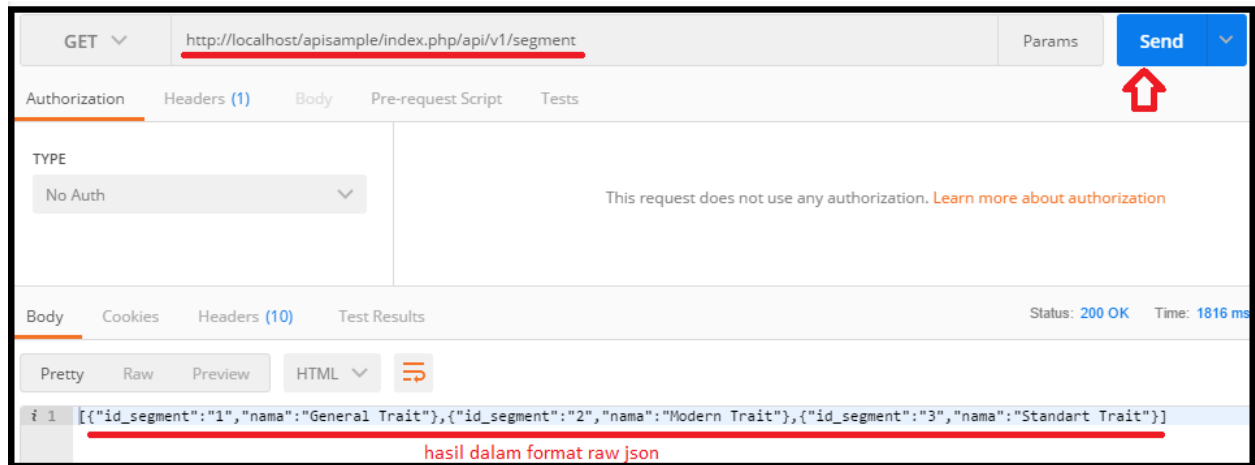
Gambar 24. Menambahkan require Koneksi.php

Langkah keempat, tambahkan pula require 'source/routes/segment.php';, didalam file index.php

```
1  <?php  
2  
3  use \Psr\Http\Message\ServerRequestInterface as Request;  
4  use \Psr\Http\Message\ResponseInterface as Response;  
5  
6  
7  require 'vendor/autoload.php';  
8  require 'source/config/Koneksi.php';  
9  
10 //instantiate the Slim app object  
11 $app = new \Slim\App;  
12  
13 /** implement rest of the methods here **/  
14  
15 //Call Routes {v1:: localhost/apisample/api/v1/[nama_route/[action]]}  
16 require 'source/routes/segment.php';  
17  
18 /** rest of the methods end here **/  
19  
20 $app->run();
```

Gambar 25. Baris Kode index.php

Langkah kelima, menguji web yang telah dibuat. Untuk proses read, langsung saja dengan membuka link yang terkait atau bisa juga dengan menggunakan postman.

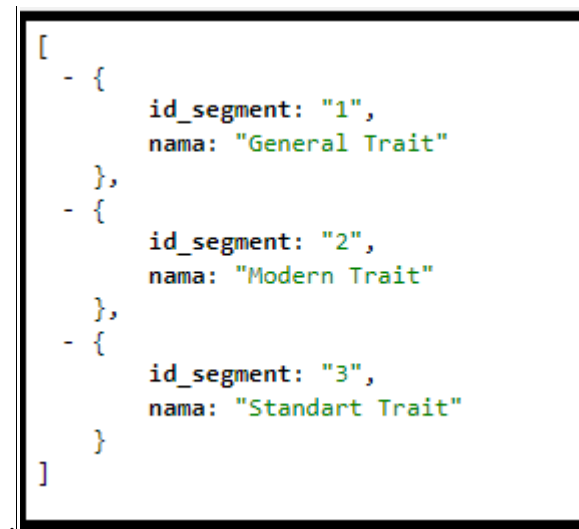


Gambar 26. Url get All data segment dengan Postman



Gambar 27. Url get All data segment

Jika proses berhasil maka browser akan menampilkan data-data yang ada didalam tabel segmen dalam format raw JSON. pada browser format tersebut dapat dirubah menjadi pretty JSON dengan menambah extensi chrome JSON View



Gambar 28. Hasil data segment dalam format pretty JSON

Langkah ke enam, meneruskan untuk proses add, edit dan delete pada segment. Berikut potongan kode untuk insert, update dan delete.

```
//insert data segment
$app->post('/api/v1/segment/add', function(Request $request, Response $response){
    //get parameter by nama in database segment
    $name = $request->getParam('nama');
    //set sql query
    $query = "INSERT INTO segment (nama) VALUES (?) ";
    //make try-catch for error handling
    try {

        //initiate object from Class Koneksi
        $dbase = new Koneksi();
        //call Method Connection in $conn
        $conn = $dbase->Connection();
        //prepare the sql query
        $state = $conn->prepare($query);
        //get data from query result
        $state->bindParam(1, $name);
        //run the sql query
        $state->execute();

        echo '{"notice":{"text":"Data Customer was Added"}}';
        //set connection to null
        $conn=null;
    }catch(PDOException $e){
        //show error messages
        echo '{"error":{"text":"' . $e->getMessage() . '"}}';
    }
});
```

Gambar 29. Baris Kode route add data segment

Berikut penjelasan potongan script diatas :

- `$app->get('/api/v1/segment/add', function(Request $request, Response $response){`,
`$app->get()` memanggil route method GET kemudian diberikan route ke
'api/v1/segment/add' dan callback yang dilewatkan dengan dua argument, yang pertama
Request \$Request adalah `Psr\Http\Message\ServerRequestInterface` objek yang
merepresentasikan request Http saat ini. Lalu yang kedua, **Response \$Response** adalah
`Psr\Http\Message\ResponseInterface` objek yang merepresentasikan
response Http saat ini.

- `$name = $request->getParam('nama');` , mengambil nilai yang diberikan pada parameter nama dan disematkan pada variable `$name`.
- `$query = "INSERT INTO segment (nama) VALUES (?) ";` , menyiapkan pernyataan SQL untuk menyimpan record kedalam table segment berdasarkan parameter nama.
- `$dbase = new Koneksi();`, membuat objek `$dbase` agar dapat mengakses kelas Koneksi.
- `$conn = $dbase->Connection();` , mengaktifkan koneksi ke database dengan pemanggilan method `Connection()` yang ada pada kelas Koneksi.
- `$state = $conn->prepare($query);` ,menyediakan pernyataan SQL yang telah di deklarasikan sebelumnya didalam variable `$query`, untuk dieksekusi nantinya oleh fungsi `execute()`.
- `$state->bindParam(1, $name);` , index 1 bertujuan untuk memberikan nilai pada parameter nama, dengan nilai yang telah diambil sebelumnya didalam variable `$name`.
- `$state->execute();` , mengeksekusi pernyataan SQL pada `$state`.
- `echo '{"notice":{"text":"Data Segment was Added"}}';` , menampilkan pesan jika berhasil menambah record data.
- `$conn = null;` , menutup koneksi ke database.
- `catch(PDOException $e)`, menampilkan kesalahan exception diikuti detail kesalahan yang terjadi pada baris berikutnya

```

//update data segment
$app->put('/api/v1/segment/edit/{id}', function(Request $request, Response $response){
    //get value has passed in {id}
    $id = $request->getAttribute('id');
    //get parameter by nama in database segment
    $nama = $request->getParam('nama');
    //set the sql query
    $query = "UPDATE segment SET nama=? WHERE id_segment = $id";
    //make try-catch for error handling
    try{

        //initiate object from Class Koneksi
        $dbase = new Koneksi();
        //call Method Connection in $conn
        $conn = $dbase->Connection();
        //prepare the sql query
        $state = $conn->prepare($query);
        //get data from query result
        $state->bindParam(1, $nama);
        //run the sql query
        $state->execute();

        echo '{"notice":{"text":"Data ke- '.$id.' was Changed"}}';
        //set connection to null
        $conn=null;
    }catch(PDOException $ex){
        echo '"error":{"text":{"'.$ex->getMessage().'"}';
    }
});

```

Gambar 30. Baris Kode route edit data segment

```
//delete data segment
$app->delete('/api/v1/segment/delete/{id}', function(Request $request, Response $response){
    //get value has passed in {id}
    $id = $request->getAttribute('id');
    //set the sql query
    $query = "DELETE FROM segment WHERE id_segment=$id ";
    //make try-catch for error handling
    try {

        //initiate object from Class Koneksi
        $dbase = new Koneksi();
        //call Method Connection in $conn
        $conn = $dbase->Connection();
        //prepare the sql query
        $state = $conn->prepare($query);
        //run the sql query
        $state->execute();

        echo '{"notice":{"text":"Data ke- '.$id.' has GONEEEE!!!"}}';
        //set connection to null
        $conn=null;
    }catch(PDOException $e){
        //show error messages
        echo '"error":{"text":{"'.$e->getMessage().'"}';
    }
});|
```

Gambar 31. Baris Kode route delete data segment

Langkah ketujuh adalah menguji web untuk proses create, update, dan delete. Proses pengujian dilakukan dengan menggunakan direct post. Proses akan lebih mudah jika membuat sebuah form dengan bahasa html. Akan tetapi kita juga bisa menggunakan Postman untuk melakukan uji dengan inputan raw Json.