**Enterprise Application Development**

**SIT780**

**Assignment 1**

Tharika Dulanjali (217405002) – 100%

Edwin John (218599279) – 100%

Praneeth Goud (217557158) – 100%

**Overview of RESTful APIs**

RESTful (Representational State Transfer) Application Programming Interface (API) Descriptive Language (DL) is a formal language designed to provide a structured description of the RESTful Web API, useful for both humans and automated machines. A REST architecture style compliant web service called REST Full Web Services (RWS) provides interoperability between computer systems on the Internet. RESTful web services allow requester systems to access and manipulate textual representations of web resources using a unified, predefined set of stateless operations. Other types of web services, such as SOAP web services, expose their own set of optional operations.

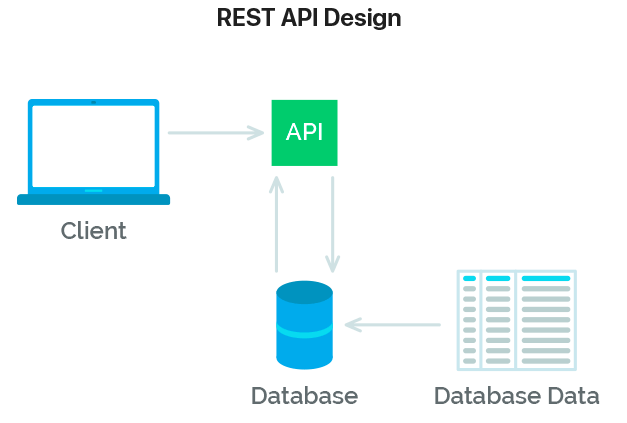
It is completely HTTP based and can therefore be used with any system or application that can send HTTP requests. "Web resources" were originally defined on the World Wide Web as documents or files identified by the URL. In the RESTful web services, requests made to the URI of a resource use HTML, XML, JSON or other payloads formatted to obtain a response.

One of the main advantages of the REST API is that they provide very high flexibility. Because data is independent of resources or methods, REST can handle multiple types of calls, return different data formats, and even make structural changes through proper implementation of hypermedia. This flexibility allows developers to build an API while meeting the needs of a variety of customers. One disadvantage of the RESTful API is that you may lose the ability to maintain REST state, such as in a session.

As the Internet industry evolves, the creation of REST APIs has become more specific and emerging best practices. Because RESTful web services do not follow standards other than HTTP, it is important to build RESTful APIs based on industry best practices to simplify development and increase client adoption. REST is a logical choice for building APIs that allow users to connect to and interact with cloud services. The RESTful API is used by sites like Amazon, Twitter, LinkedIn and Google.

**How RESTful APIs work**

The RESTful API explicitly uses the HTTP method defined in the RFC 2616 protocol. They use GET to retrieve resources. PUT changes or updates the state of a resource, such as an object, file, or block. POST creates the resource and DELETE to remove it.



RESTful API best practices come down to four essential operations and each operation uses its own HTTP methods.

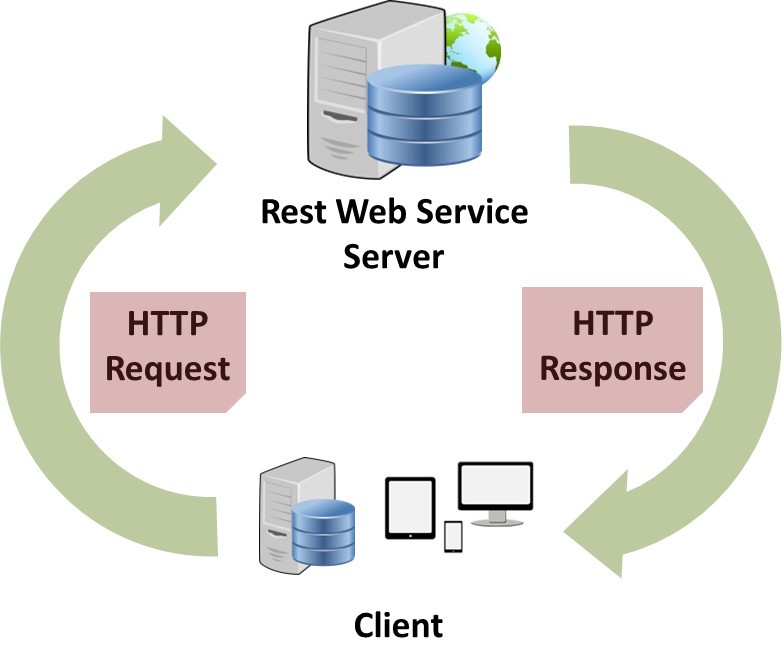
* Receiving data in a convenient format - GET
* Creating new data - POST
* Updating data - PUT
* Deleting data - DELETE

The fact that REST contains a single common interface for requests and databases is its great advantage. This can be viewed in the table below.

**REST API Architectural Constraints**

There are six architectural limitations that make any web service as described below. The only optional constraint for the REST architecture is on-demand code. If a service violates other constraints, it cannot be strictly called RESTful.

* **Client-Server**: This constraint is based on the concept that the client and server should be separated from each other and allowed to evolve separately. REST applications should have a client-server architecture. The client is requesting resources and does not care about data storage. The data storage is still internal to each server. The server is the person who holds the resources but does not care about the user interface or user status. The client does not need to know anything about the business logic, nor does the server need to know anything about the front-end UI.



* **Stateless**: The REST API is stateless, which means that calls can be made independently of each other, and each call contains all the data needed to successfully complete itself. This means that the necessary state to process the request is contained in the request itself, and the server does not store anything related to the session.
* **Cache**: Because stateless APIs can increase request overhead by handling a large number of incoming and outgoing calls, the REST API should be designed to encourage storage of cacheable data. Each response should include whether the response is cacheable and the duration of the response that can be cached on the client. The client will return any subsequent requested data from its cache and will not need to send the request to the server again.
* **Uniform Interface:** The key to decoupling the client and server is to have a unified interface that allows the application to evolve independently without the need to tightly couple the application's services or models and operations to the API layer itself. It is a key constraint that distinguishes between REST APIs and non-REST APIs. It suggests that there should be a uniform way to interact with a given server, regardless of device or application type (website, mobile app)
* **Layered System**: REST APIs have different architectural layers that work together to build hierarchies to help create more scalable and modular applications. The application architecture needs to consist of multiple layers. Except for the direct layer, each layer does not know any information about any layer, there may be many intermediate servers between the client and the end server. Intermediate servers can increase system availability by enabling load balancing and providing shared cache.
* **Code on Demand**: This is an optional feature. Code on Demand allows code or applets to be transferred through the API for use in applications. Accordingly, the server can also provide executable code to the client. Examples of on-demand code can include compiled components such as Java applets and client-side scripts, such as JavaScript.

In REST, the client must contain all the information of the server to satisfy the request, either as a query parameter, as part of a header or URI. Stateless can achieve higher availability because the server does not have to maintain, update or communicate the state of the session. There is a disadvantage when the client needs to send too much data to the server, so it reduces the scope of network optimization and requires more bandwidth.

**Rules of REST API:** There are some rules to keep in mind when creating REST API endpoints.

* REST is based on the resources or nouns rather than actions or verbs. This means that the URI of the REST API should always end with a noun.
* HTTP predicates are used to identify actions. Some HTTP verbs are – GET, PUT, POST, DELETE, UPDATE, PATCH.
* Web applications should be organized into resources such as users, and then modified using HTTP verbs (such as GET, PUT, POST, DELETE).

**HTTP verbs:** Some common HTTP methods/verbs are described below:

* **GET:** Retrieve one or more resources identified by the request URI, which can buffer the receipt of information.
* **POST:** The resource is created when the request is submitted, in which case the response is not cacheable. This method is not secure if security is not applied to the endpoint because it allows anyone to create random resources through submission.
* **PUT:** Update the existing resources on the server specified by the request URI.
* **DELETE:** Delete the existing resources on the server specified by the request URI. It always returns the appropriate HTTP status for each request.



**Implementation of Rest API in PHP**

**Setup the database**

**Certain critical steps need to be carefully followed while setting up the database. We first created a file ‘assignment1.txt’ to create a table with all the basic details like ‘First name’, ‘surname’,’email’,’sid’ and also all the student values which added initial values to it.**

**The version of Putty we used was ‘0.70.0.0’ and using putty, we connected to the Deakin’s database using the Host name ‘interactive.deakin.edu.au’ and logged in with user details.   
Once the putty’s connection has been successfully established, we now need to connect to SQLplus to SSID and enter the SQL password. Now this will prompt us to create a table, once the required table is created and the values are added, we can now proceed to the implementation of Representational State Transfer (REST) in PHP which as follows.**

**Create Students Table**

create table tbl\_students(

sid INTEGER,

fname varchar2(100),

sname varchar2(100),

email varchar2(100)

);

**Dump data for Students Table**

insert into tbl\_students values (1, 'John', 'Grundy', 'john@gmail.com');

insert into tbl\_students values (2, 'Sam', 'Goodman', 'sam@yahoo.com');

insert into tbl\_students values (3, 'Joe', 'Griffith', 'joe.griffith@gmail.com');

insert into tbl\_students values (4, 'Monica', 'Lim', 'monica@deakin.edu.au');

**Create Operation**

* **Open api folder and then create a new create.php file**
* **Create a function called create() which takes the student id, firstname, surname and email id as parameters**
* **The function establishes a connection to the database using the appropriate credentials and adds a new row with the given values using a SQL query**
* **The values are passed to create.php using get method from the client side**

**Student create() method**

$dbuser="xxxxx"; //Deakin username

$dbpass="yyyyyy"; //Deakin password

$db="SSID"; //the Deakin database is by default SSID

$connection=ocilogon($dbuser, $dbpass, $db);

if(!$connection)

{

echo("An error occurred connecting to the database");

exit;

}

function create($SID, $FNAME, $SNAME, $EMAIL){ //receive values as input

//query code

$EMAIL = (string) $EMAIL;

$query = "INSERT INTO tbl\_students VALUES($SID, '$FNAME', '$SNAME', '$EMAIL')"; //changed query to insert and provided values

$stmt=ociparse($GLOBALS['connection'], $query);

if(!$stmt)

{

echo("An error occurred in parsing the SQL string. \n");

exit;

}

ociexecute($stmt);

}

$SID = $\_GET['sid'];

$FNAME = $\_GET['fname'];

$SNAME = $\_GET['sname'];

$EMAIL = $\_GET['email'];

create($SID, $FNAME, $SNAME, $EMAIL);

Output

To test for successful creation, open browser. Enter the following as the request URL

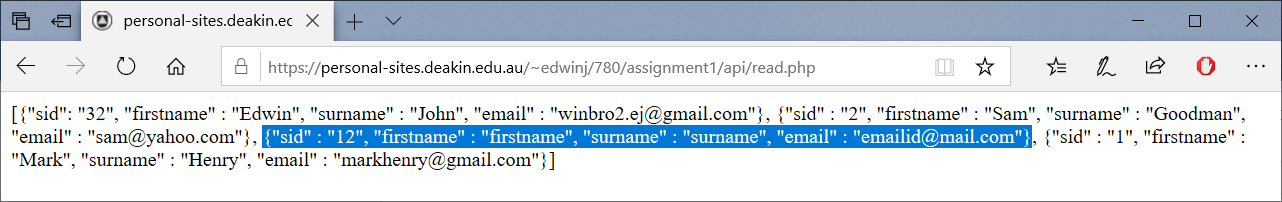
<https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/create.php?sid=12&fname=firstname&sname=surname&email=emailid@mail.com>

Then enter the following url to check if new values have been created or not:

<https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/read.php>

You’ll get the following JSON added to existing data:

[{"sid" : "12", "firstname" : "firstname", "surname" : "surname", "email" : "emailid@mail.com"}]



**Read Operation**

* **Create read.php inside api folder**
* **Create a function called read() which doesn’t take any parameters**
* **The function establishes** a connection to the database using the appropriate credentials and gets all the rows of the table and echos it out in JSON format.

**Student read() method**

$dbuser="xxxxx"; //Deakin username

$dbpass="yyyyy"; //Deakin password

$db="SSID"; //the Deakin database is by default SSID

$connection=ocilogon($dbuser, $dbpass, $db);

if(!$connection)

{

echo("An error occurred connecting to the database");

exit;

}

function read(){

//query code

$query="SELECT \* FROM tbl\_students";

$stmt=ociparse($GLOBALS['connection'], $query);

if(!$stmt)

{

echo("An error occurred in parsing the SQL string. \n");

exit;

}

ociexecute($stmt);

//query code ends

$table=array();

//fetching code

while(ocifetch($stmt)){ //fetches a row of data... and the data is in stmt

$row = array();

$fg1=ociresult($stmt,"SID");

$fg2=ociresult($stmt,"FNAME");

$fg3=ociresult($stmt,"SNAME");

$fg4=ociresult($stmt,"EMAIL");

$row=array($fg1,$fg2,$fg3,$fg4);

array\_push($table,$row);

}

return $table;

}

$table = read();

//traversing and rendering(plain JSON):

$r=count($table);

$c=count($table[0]);

echo "[";

for($n=0;$n<$r;$n++){

echo "{";

for($m=0;$m<$c;$m++){

if($m%4==0)

echo "\"sid\" : ". "\"".$table[$n][$m]."\", ";

else if($m%4==1)

echo "\"firstname\" : ". "\"".$table[$n][$m]."\", ";

else if($m%4==2)

echo "\"surname\" : ". "\"".$table[$n][$m]."\", ";

else if($m%4==3)

echo "\"email\" : ". "\"".$table[$n][$m]."\"";

}

if($n==$r-1)

echo "}";

else

echo "}, ";

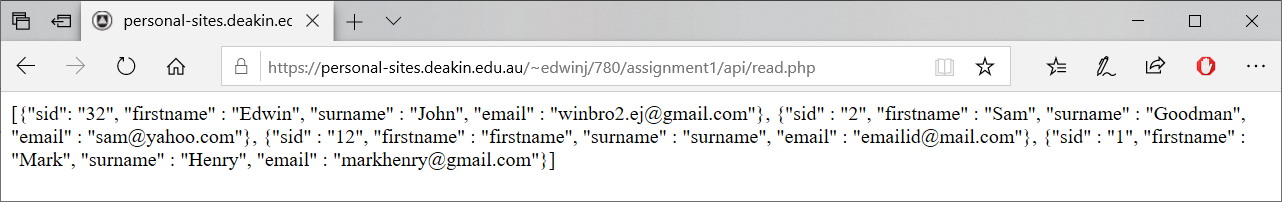
}

echo "]";

Output

To test for successful creation, open browser. Enter the following as the request URL

<https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/read.php>



**Update Operation**

* **Create update.php inside api folder**
* **Create a function called update() which takes the student id, firstname, surname and email id and also the student id of the student which requires the changes.**
* **The function establishes a connection to the database using the appropriate credentials and updates the row with the new details.**
* **The values are passed to update.php using get method from the client side.**

**Student update() method**

$dbuser="xxxxx"; //Deakin username

$dbpass="yyyyy"; //Deakin password

$db="SSID"; //the Deakin database is by default SSID

$connection=ocilogon($dbuser, $dbpass, $db);

if(!$connection)

{

echo("An error occurred connecting to the database");

exit;

}

function update($updateSID, $SID, $FNAME, $SNAME, $EMAIL){ //receive values as input

//query code

$EMAIL = (string) $EMAIL;

$query = "UPDATE tbl\_students SET SID=$SID, FNAME='$FNAME', SNAME='$SNAME', EMAIL='$EMAIL' WHERE SID=$updateSID"; //changed query to insert and provided values

$stmt=ociparse($GLOBALS['connection'], $query);

if(!$stmt)

{

echo("An error occurred in parsing the SQL string. \n");

exit;

}

ociexecute($stmt);

}

$updateSID = $\_GET['updatesid'];

$SID = $\_GET['sid'];

$FNAME = $\_GET['fname'];

$SNAME = $\_GET['sname'];

$EMAIL = $\_GET['email'];

update($updateSID, $SID, $FNAME, $SNAME, $EMAIL);

Output

**Open browser. Enter the following as the request URL.**

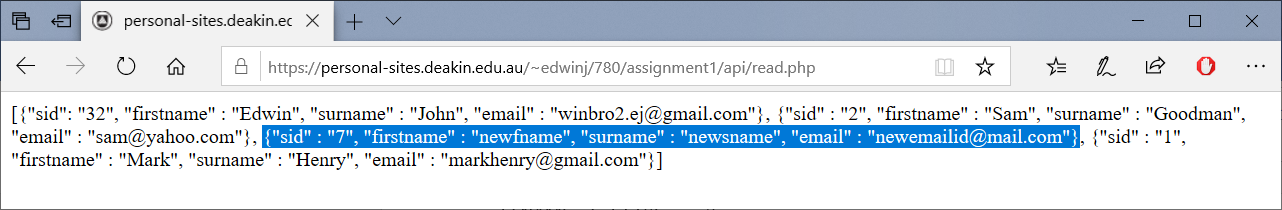
<https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/update.php?updatesid=12&sid=7&fname=newfname&sname=newsname&email=newemailid@mail.com>

Then enter the following url to check if the values have been updated or not:

<https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/read.php>

You’ll get the following JSON replacing the particular data:

[{"sid" : "7", "firstname" : "newfname", "surname" : "newsname", "email" : "newemailid@mail.com"}]



**Delete Operation**

* Create delete.php inside api folder
* Create a function called delete() which takes the student id as parameters
* The function establishes a connection to the database using the appropriate credentials and deletes the row with the given student id
* The values are passed to delete.php using get method from the client side

**Student delete() method**

$dbuser="xxxxx"; //Deakin username

$dbpass="yyyyy"; //Deakin password

$db="SSID"; //the Deakin database is by default SSID

$connection=ocilogon($dbuser, $dbpass, $db);

if(!$connection)

{

echo("An error occurred connecting to the database");

exit;

}

function delete($SID){ //receive input as SID

//query code

$query = "DELETE FROM tbl\_students where SID=$SID"; //changed query to delete based on SID

$stmt=ociparse($GLOBALS['connection'], $query);

if(!$stmt)

{

echo("An error occurred in parsing the SQL string. \n");

exit;

}

ociexecute($stmt);

}

$SID = $\_GET['sid'];

delete($SID);

Output

**Open browser. Enter the following as the request URL.**

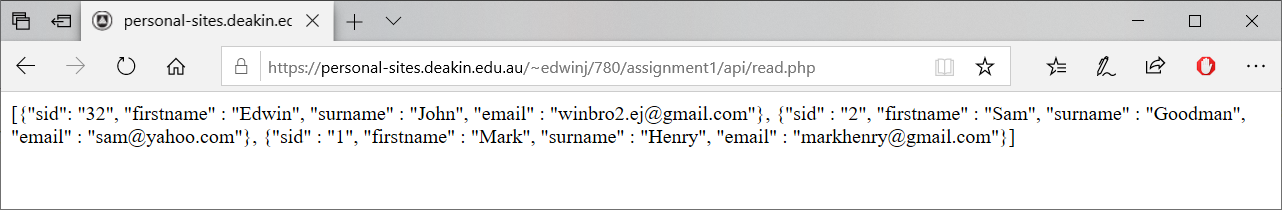
[https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/delete.php? sid=7](https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/delete.php?%20sid=7)

Then enter the following url to check if the values have been deleted or not:

<https://personal-sites.deakin.edu.au/~edwinj/780/assignment1/api/read.php>

You’ll observe that the following JSON will not be there anymore:

[{"sid" : "7", "firstname" : "newfname", "surname" : "newsname", "email" : "newemailid@mail.com"}]



**Sending API requests using CURL**

Firstly, we created index.html from which the user or the client inputs the data and performs CRUD operations. Every time an operation is performed, index.html sends a GET request to the php file associated with that operation.

In the php file, CURL is initialized, its parameters are set and finally it is executed. The URL of our api is one of the parameters. After the result is obtained, the curl channel is closed.  
  
Hence, CURL will retrieve the result of the operation from the API. The browser’s job is to make that code visually readable. Curl shows you what you’re actually retrieving. This is how the CURL is used to consume the API.

# **Sending a CREATE request**

//**initialize curl**

$ch = curl\_init();

//**this is the api url for create method.**

$url = "https://personal-sites.deakin.edu.au/~edwinj/780/as1/api/create.php?sid=$sid&fname=$fname&sname=$sname&email=$email";

//**Sets the url an option on the given cURL session handle.**

curl\_setopt($ch, CURLOPT\_URL, $url);

// **Set so curl\_exec returns the result instead of outputting it.**

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, true);

// **Get the response and close the channel.**

$response = curl\_exec($ch);

curl\_close($ch);

# **Sending a READ request**

//**creating a table**

echo "<table >

<tr>

<th>SID</th>

<th>Firstname</th>

<th>Lastname</th>

<th>Email</th>

</tr>

";

//**api url for the read operation**

$url = "https://personal-sites.deakin.edu.au/~edwinj/780/as1/api/read.php";

//**initialize curl**

$ch = curl\_init();

//**Sets the url as an option on the given cURL session handle.**

curl\_setopt($ch, CURLOPT\_URL, $url);

// **Set so curl\_exec returns the result instead of outputting it.**

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, true);

// **Get the response and close the channel.**

$response = curl\_exec($ch);

curl\_close($ch);

//**decode the JSON result into an array and then echo it out in the table**

$table = json\_decode($response,true);

$r=count($table);

$c=count($table[0]);

for($n=0;$n<$r;$n++){

echo "<tr>";

echo "<td>".$table[$n]["sid"]."</td>";

echo "<td>".$table[$n]["firstname"]."</td>";

echo "<td>".$table[$n]["surname"]."</td>";

echo "<td>".$table[$n]["email"]."</td>";

echo "</tr>";

}

echo "</table>";

# **Sending an UPDATE request**

//**api url for update operation**

$url = "https://personal-sites.deakin.edu.au/~edwinj/780/as1/api/update.php?updatesid=$updatesid&sid=$sid&fname=$fname&sname=$sname&email=$email";

//**initialize curl**

$ch = curl\_init();

//**Sets the url as an option on the given cURL session handle.**

curl\_setopt($ch, CURLOPT\_URL, $url);

// **Set so curl\_exec returns the result instead of outputting it.**

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, true);

// **Get the response and close the channel.**

$response = curl\_exec($ch);

curl\_close($ch);

# **Sending a DELETE request**

//**api url for delete operation**

$url = "https://personal-sites.deakin.edu.au/~edwinj/780/as1/api/delete.php?sid=$sid";

//**initialize curl**

$ch = curl\_init();

//**Sets an option on the given cURL session handle.**

curl\_setopt($ch, CURLOPT\_URL, $url);

// **Set so curl\_exec returns the result instead of outputting it.**

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, true);

// **Get the response and close the channel.**

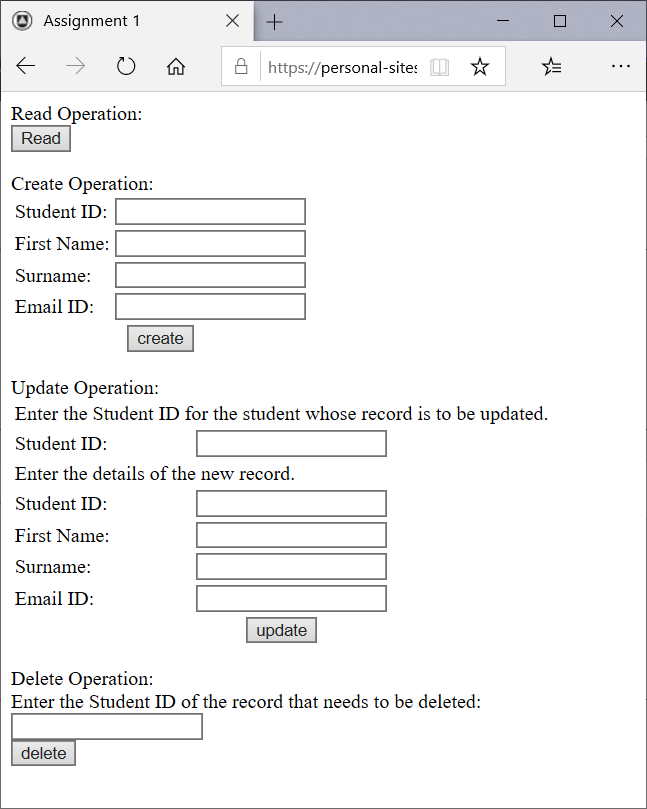
$response = curl\_exec($ch);

curl\_close($ch);

**OUTPUTS of any use case:**

Let’s say the user has to read the existing values in database.

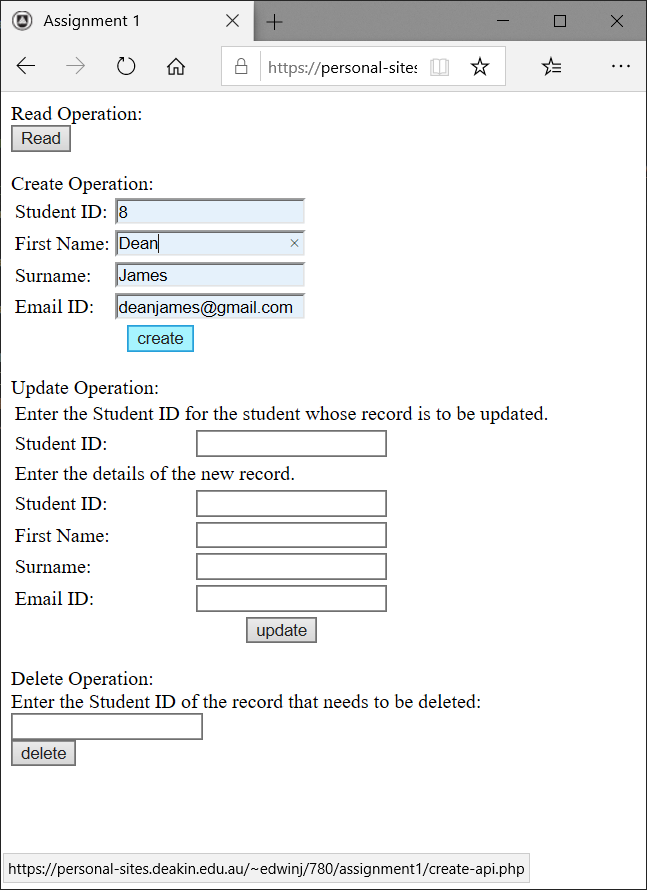
Index.html:



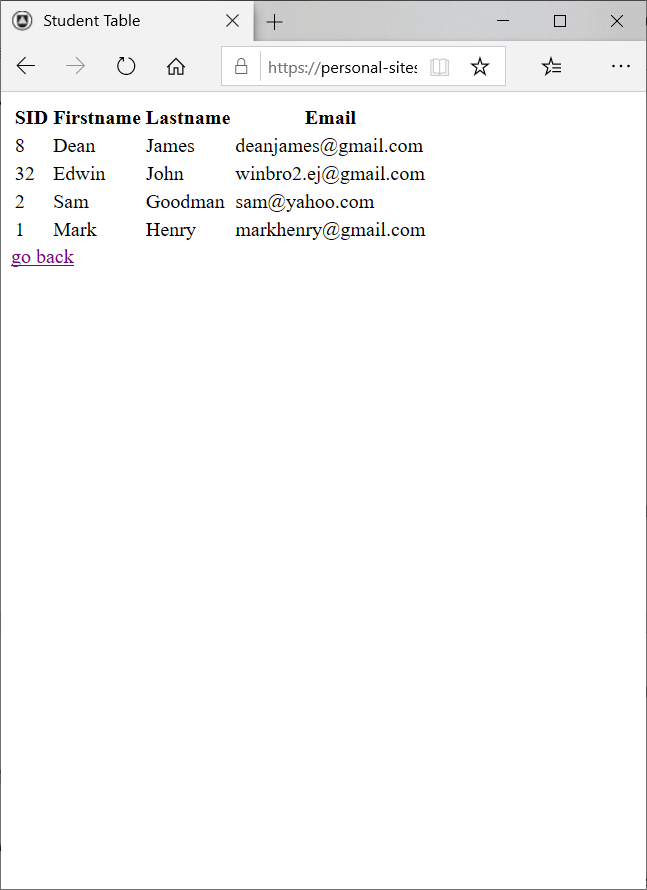
After clicking on read:



Now, if we want to add new values, click on go back, enter the required values and press create.



Then we can click on Read button to show the table with the new values.



Similarly, other operations may also be performed.

**References**

*Filipe Ximenes, F. (2019). How to design a RESTful API architecture from a human-language spec. [online] O'Reilly Media. Available at: https://www.oreilly.com/learning/how-to-design-a-restful-api-architecture-from-a-human-language-spec [Accessed 10 March. 2019]*

*Toptal Engineering Blog. (2019). Building REST API for Legacy PHP Projects. [online] Available at: https://www.toptal.com/php/building-rest-api-for-legacy-php-projects [Accessed 13 March. 2019]*

*The Code of a Ninja. (2019). How To Create A Simple REST API in PHP - Step By Step Guide!. [online] Available at: https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html [Accessed 27 March. 2019]*

*Mlsdev.com. (2019). {{global.page.social.title}}. [online] Available at: https://mlsdev.com/blog/81-a-beginner-s-tutorial-for-understanding-restful-api [Accessed 1 Apr. 2019]*

*IBM Developer. (2019). RESTful Web services. [online] Available at: https://developer.ibm.com/articles/ws-restful/ [Accessed 2 Apr. 2019]*

*Docs.oracle.com. (2019). Sending API requests using cURL. [online] Available at: https://docs.oracle.com/cloud/latest/marketingcs\_gs/OMCAB/Developers/GettingStarted/API%20requests/curl-requests.htm [Accessed 3 Apr. 2019]*

*Weichler, B. (2019). cURL API calls with PHP and json data (GET, POST, PUT, DELETE). [online] Bob Weichler. Available at: https://www.weichieprojects.com/blog/curl-api-calls-with-php/ [Accessed 4 Apr. 2019]*

*Docs.oracle.com. (2019). Sending API requests using cURL. [online] Available at: https://docs.oracle.com/cloud/latest/marketingcs\_gs/OMCAB/Developers/GettingStarted/API%20requests/curl-requests.htm [Accessed 5 Apr. 2019]*