CSE 3101: Internet Computing II

Lab Session 6

Paul Crawford

Semester 1, Week 11 (5th & 6th November, 2018)

1. Aims

1.1. Further understanding of Web applications, including client-side Web Forms (with server-side processing upon submission), server-side PHP functions, and Web Service architectures.

1.2. Further increased facility with the ongoing sets of concepts & techniques needed for eventual completion of the Final Project.

1.3. **POLICY CHANGE (REMINDER)**: Please note that henceforth **all Lab Sessions will be marked** and count as a portion of the overall Lab grade (10%). Completed labwork must be zipped into an archive (filename given later below at the end of § 2.1), and submitted to the Tutor (<pcrawford@mac.com>) by midnight Friday each week, with the Subject line: 'CSE3101 - Lab <*N*> - <*YourGroupName*>'.

2. Tasks

2.1. Web API (RESTful Flavour), Continued

2.1.1. Continue to use the RESTful API folder (kindly provided by the 2nd Lecturer) from Lab Session 5 {whose materials are still available at the tutor's Nyanza Software website (<www.nyanzasoftware.com>) > 'Teaching', § 'University of Guyana' > 'CSE3101: Internet Computing II'} — specifically, its extracted 'restapi' subfolder that was copied into your Apache Web Root directory (e.g., 'C:\xampp\htdocs').

2.1.2. Continue to use the test database named 'api_db' that was set up in Lab Session 5 via the PhpMyAdmin console.

2.1.3. Continue to exercise the Web API functionality (the 'R' or query URIs can be tested via any Web browser, and the 'U' URI can be tested via either the Tutor's extra web form or a specialised web service such as Postman), and save the various **modified source files** and **result files** as follows:

2.1.3.1. Navigate to 'localhost/restapi/**product/read_one.php?id=28**' {to retrieve the existing 'Wallet (Improved)' product already updated in Lab Session 5}, and save the resulting JSON data (there should be just 1 entry) to a file named 'Product_Read_One_**Original**.json'.

2.1.3.2. **Modify** the 'C:\xampp**restapi\product\update.php**' PHP source file, to allow **omitting** any three of the four non-'id' fields (from the decoded JSON submitted data), where on the server side "omitting" is signified by an **empty** PHP value (i.e., *undefined*/NULL, an empty-string or the number 0).

I.e., it must be possible to submit (from the client side) an update request that specifies just the 'id' field and any **subset** of the remaining four fields (rather than requiring all four to be filled-in to avoid accidentally erasing them).

Note that you could use either the '`if`' statement or the ternary conditional operator (`?:`) to guard against referencing an empty field-value (object-member).

A useful built-in predicate to test for empty values is the aptly-named: **`empty()`**.

For example, we could adjust the assignments of the decoded JSON `$data` object's fields to the `$product` object's corresponding fields, as follows (where changes are indicated in **bold**):

```
[…]

// get id of product to be edited
$data = json_decode(file_get_contents("php://input"));

// set ID property of product to be edited
$product->id = $data->id;

// set product property values
$product->name = ( ! empty( $data->name ) ? $data->name : "" );
$product->price = ( ! empty( $data->price ) ? $data->price : "" );
$product->description = …;  // Do likewise
$product->category_id = …;  // Do likewise

[…]
```

2.1.3.3. **Also modify** the 'C:\xampp**restapi\objects\product.php**' PHP source file, '**function update()**', for all the abovementioned reasons. {Remember though, in this workhorse function/method, to verify that **at least one** non-'id' field is non-empty!}

Thus, the corresponding SQL 'UPDATE …' command must **not** set any omitted fields, i.e.: it must **set only non-empty fields**.

Note that you could use the string-concatenation operator (`.`), and the ternary conditional operator (`?:`), to build up the command's 'SET …' clause so that it includes only those fields with non-empty values, and has separating commas only where necessary.

Again, a useful built-in predicate to test for empty values is the aptly-named: **`empty()`**.

Be especially **careful with those separating commas**; recall that a comma is needed only when there's a field expression **both before and after** it!

Also, for the later prepared SQL statement, **only the non-empty fields must get bound as Params**.

For example, we could apply all of the abovementioned adjustments, as follows (where changes are indicated in **bold**):

```
[…]

// update the product
function update(){

    // sanitize
    //
    // [NOTE: This existing section was moved from below up to here]
    //
    $this->name=htmlspecialchars(strip_tags($this->name));
    $this->price=htmlspecialchars(strip_tags($this->price));
    $this->description=htmlspecialchars(strip_tags($this->description));
    $this->category_id=htmlspecialchars(strip_tags($this->category_id));
    $this->id=htmlspecialchars(strip_tags($this->id));

    // (non-)emptiness tests
    $bNameSupplied = ! empty( $this->name );
    $bPriceSupplied = ! empty( $this->price );
    $bDescSupplied = …;  // Do likewise
    $bCatIDSupplied = …;  // Do likewise

    // Verify that at least one non-'id' field was supplied
    //
    // [NOTE: Fill in the missing flags where indicated below ('…')!]
    //
    if ( ! ( $bNameSupplied || $bPriceSupplied || … || … ) ) {
        return false;
    }

    // update query
    //
    // [NOTE: Fill in the missing flags where indicated below ('…')!]
    //
    // [NOTE: It's also possible to simplify each comma-expression by
    //  appending it to the then part ('?') of the preceding field-&-
    //  placeholder-expression, and eliminating its condition-test's
    //  LHS – i.e., the flag to the left of the '&&' (and that '&&'
    //  itself) – as that LHS is already satisfied. These 3 optional
    //  simplifications are left as an exercise for the reader. :-)]
    //
    $query = "UPDATE
                " . $this->table_name . "
            SET
               " . ( $bNameSupplied ? "name = :name" : "" ) .
                     ( ( $bNameSupplied &&
                         ( $bPriceSupplied || $bDescSupplied ||
                           $bCatIDSupplied ) )
                       ? "," : "" ) . "
               " . ( $bPriceSupplied ? "price = :price" : "" ) .
                     ( ( $bPriceSupplied &&
                         ( $bDescSupplied || $bCatIDSupplied ) )
                       ? "," : "" ) . "
               " . ( … ? "description = :description" : "" ) .
                     ( ( … && $bCatIDSupplied ) ? "," : "" ) . "
               " . ( … ? "category_id = :category_id" : "" ) . "
            WHERE
                id = :id";
```

```
        // prepare query statement
        $stmt = $this->conn->prepare($query);

        // bind new values
        //
        // [NOTE: Fill in the missing flags where indicated below ('…')!]
        //
        if ( $bNameSupplied ) { $stmt->bindParam(':name', $this->name); }
        if ( $bPriceSupplied ) { $stmt->bindParam(':price', $this->price); }
        if ( … ) $stmt->bindParam(':description', $this->description); }
        if ( … ) $stmt->bindParam(':category_id', $this->category_id); }

        […]
    }

    […]
```

2.1.3.4. Navigate to 'localhost/restapi/**paulc_extras/cud_client.html**' OR use a web service such as Postman, and update the existing 'Wallet (Improved)' product (already modified in Lab Session 5) by submitting to the URI 'localhost/restapi/**product/update.php**' the following POST data in JSON format, this time updating the **name** only:

For the Client Form (which will auto-convert into JSON):
id : 28
name : Wallet **(New and Improved)**
(Note: All other fields are to be left **empty**.)

For Postman & similar services (making sure to select the JSON data-type):

```
{
    "id"   : 28,
    "name" : "Wallet (New and Improved)"
}
```

Make sure that you see the result: 'message: Product was updated.'. {Note: For the Client Form, it will be logged to the Console.}

2.1.3.5. (Re-)Navigate to 'localhost/restapi/**product/read_one.php?id=28**'. Verify that the **name** was indeed correctly updated, and also that all other fields still retain their existing values (i.e., have not been erased or set to null, "" or 0). Save the resulting JSON data (there should of course still be just 1 entry) to a file named 'Product_Read_One_AfterU_**Name**.json'.

2.1.3.6. (Re-)Navigate to 'localhost/restapi/**paulc_extras/cud_client.html**' OR use a web service such as Postman, and update the existing 'Wallet (Improved)' product (already modified in Lab Session 5) by submitting to the URI 'localhost/restapi/**product/update.php**' the following POST data in JSON format, this time updating the **description** only:

For the Client Form (which will auto-convert into JSON):
id : 28

description : You can **really** use this one!
(Note: All other fields are to be left **empty**.)

For Postman & similar services (making sure to select the JSON data-type):

```
{
    "id"          : 28,
    "description" : "You can really use this one!"
}
```

Make sure that you see the result: 'message: Product was updated.'. {Note: For the Client Form, it will be logged to the Console.}

2.1.3.7. (Re-)Navigate to 'localhost/restapi/**product/read_one.php?id=28**'. Verify that the **description** was indeed correctly updated, and also that all other fields still retain their existing values (i.e., have not been erased or set to null, "" or 0). Save the resulting JSON data (there should of course still be just 1 entry) to a file named 'Product_Read_One_AfterU_**Description**.json'.

2.1.3.8. (Re-)Navigate to 'localhost/restapi/**paulc_extras/cud_client.html**' OR use a web service such as Postman, and update the existing 'Wallet (Improved)' product (already modified in Lab Session 5) by submitting to the URI 'localhost/restapi/**product/update.php**' the following POST data in JSON format, this time updating the **price** only:

For the Client Form (which will auto-convert into JSON):
id : 28
price : **12**99
(Note: All other fields are to be left **empty**.)

For Postman & similar services (making sure to select the JSON data-type):

```
{
    "id"    : 28,
    "price" : 1299
}
```

Make sure that you see the result: 'message: Product was updated.'. {Note: For the Client Form, it will be logged to the Console.}

2.1.3.9. (Re-)Navigate to 'localhost/restapi/**product/read_one.php?id=28**'. Verify that the **price** was indeed correctly updated, and also that all other fields still retain their existing values (i.e., have not been erased or set to null, "" or 0). Save the resulting JSON data (there should of course still be just 1 entry) to a file named 'Product_Read_One_AfterU_**Price**.json'.

2.1.3.10. (Re-)Navigate to 'localhost/restapi/**paulc_extras/cud_client.html**' OR use a web service such as Postman, and update the existing 'Wallet (Improved)' product (already modified in Lab Session 5) by submitting to the URI 'localhost/restapi/

product/update.php' the following POST data in JSON format, this time updating the **category_id** only:

For the Client Form (which will auto-convert into JSON):
id : 28
category_id : **1**
(Note: All other fields are to be left **empty**.)

For Postman & similar services (making sure to select the JSON data-type):

```
{
    "id"          : 28,
    "category_id" : 1
}
```

Make sure that you see the result: 'message: Product was updated.'. {Note: For the Client Form, it will be logged to the Console.}

2.1.3.11. (Re-)Navigate to 'localhost/restapi/**product/read_one.php?id=28**'. Verify that the **category_id** was indeed correctly updated, and also that all other fields still retain their existing values (i.e., have not been erased or set to null, "" or 0). Ssave the resulting JSON data (there should of course still be just 1 entry) to a file named 'Product_Read_One_AfterU_**Category_ID**.json'.

2.1.3.12. From the above **2 modified source files** and **5 result files** (**7 files overall**) — i.e.: '**update.php**', '**product.php**', 'Product_Read_One_Original.json', 'Product_Read_One_AfterU_Name.json', 'Product_Read_One_AfterU_Description.json', 'Product_Read_One_AfterU_Price.json', and 'Product_Read_One_AfterU_Category_ID.json' — create a zip archive named 'CSE3101_Lab06_<*YourGroupName*>.zip' and submit it as described in § 1.3.

2.2. Overall Considerations

2.2.1. For additional information, practice and code samples, you could also explore any of various useful articles, tutorials & references available online. For instance, the **Wikipedia** site (<en.wikipedia.org>) has several detailed articles on Web Services, Web APIs, the REST architecture, etc.