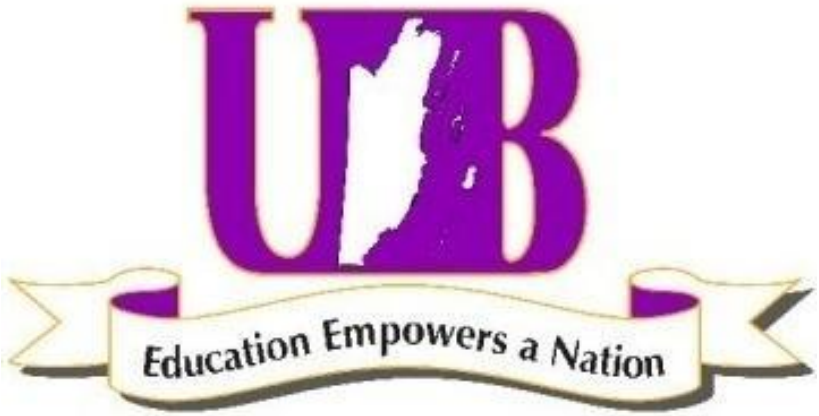

Software Engineering

CMPS 4131

Victor Tillett,Daniel Garcia,Azriel Cuellar, LEVI COC ,Floyd Ack

Project 2 Technical Documentation

March 30th, 2023

CODE DOCUMENTATION

Booking History(bookinghistory.php)

This is a PHP function named bookinghistory() that retrieves booking history for a given ‘userid’ parameter.

Here is a detailed explanation of the code:

This line declares the function ‘bookinghistory’ which accepts a parameter ‘$userid’ to retrieve the booking history of that user.

```
function bookinghistory($userid)
```

This line declares a SQL query to retrieve the booking history for the given ‘$userid’. It selects the ticket ID, first name of the user, seat type, beginning location, bus departure time, and cost. It uses several JOIN statements to connect various tables in the database and fetch the required information.

```
    $seltrip="SELECT T.ticketid,
UI.fname,S.seattype,LS.location,BS.busdepart,BS.cost
    From ticket T
    JOIN busschedule as BS
    ON BS.busscheid = T.busscheid
    JOIN locationofstop AS LS
    ON BS.begin_locationid = LS.stoplocid
    JOIN locationofstop AS L
    ON BS.end_locationid = L.stoplocid
    JOIN seat AS S ON T.seatid = S.seatid
    JOIN userinfo as UI
    ON T.userid = UI.userid
    WHERE UI.userid = ?
```

These lines prepare the SQL query, bind the $userid parameter to the query, and execute it. The get_result() function retrieves the result set from the executed query.

```
$queryrun=mysqli_prepare($conn,$seltrip);
    $queryrun->bind_param('s', $userid);
    $queryrun->execute();
    $result=$queryrun->get_result();
```

If no records are found in the database for the given $userid, this code block will execute and display a message stating that no booking history is found.

```
    if(mysqli_num_rows($result)==0){
        echo "No Book History With This User Found";
```

If records are found in the database for the given $userid, this code block will execute and loop through each row of the result set. It will display the ticket ID, user's first name, seat type, beginning location, bus departure time, and cost for each record found.

```
lse{

        while($row = mysqli_fetch_assoc($result)){
            echo "Ticket ID: " . $row["ticketid"] . "<br>";
            echo "first Name: " . $row["fname"] . "<br>";
            echo "Seat Name: " . $row["seattype"] . "<br>";
            echo "Begin Location: " . $row["location"] . "<br>";
            echo "Begin depart: " . $row["busdepart"] . "<br>";;
            echo "Cost: " . $row["cost"] . "<br>";
        }
    }
```

Check Payment(checkpaymentinfo.php)

This line of code defines the SQL query to retrieve the payment information for the given user ID. It joins several tables and selects specific columns to retrieve.

```
    $query = "SELECT T.ticketid,
UI.fname,S.seattype,LS.location,BS.busdepart,BS.cost
    From ticket T
    JOIN busschedule as BS
    ON BS.busscheid = T.busscheid
    JOIN locationofstop AS LS
    ON BS.begin_locationid = LS.stoplocid
    JOIN locationofstop AS L
    ON BS.end_locationid = L.stoplocid
    JOIN seat AS S ON T.seatid = S.seatid
    JOIN userinfo as UI
    ON T.userid = UI.userid
    WHERE UI.userid = ?";
```

This line of code prepares the SQL statement by creating a prepared statement object from the database connection and the SQL query.

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

This line of code binds the user ID parameter to the prepared statement object.

```
$stmt->bind_param("i", $userid);
```

This line of code executes the prepared statement.

```
$stmt->execute();
```

This line of code gets the result set from the executed prepared statement.

```
$result = $stmt->get_result();
```

This line of code executes if there is no rows found with the information we are looking for.

```
if ($result->num_rows === 0) {
        echo "No Payment Info Found";
    }
```

This line of code displays the payment information for the user

```
 while ($row = $result->fetch_assoc()) {
          echo "Ticket ID: " . $row["ticketid"] . "<br>";
          echo "first Name: " . $row["fname"] . "<br>";
          echo "Seat Name: " . $row["seattype"] . "<br>";
          echo "Begin Location: " . $row["location"] . "<br>";
          echo "Begin depart: " . $row["busdepart"] . "<br>";;
          echo "Cost: " . $row["cost"] . "<br>";
       }
```

Edit User info (edituserinfo.php)

This is a PHP function named ‘edithprofile()’.

It accepts five parameters: $userid, $fname, $lname, $phone, and $address.

The require statement includes a file called connection.php, which presumably contains the code for connecting to a MySQL database.

```
<?php

function edithprofile($userid,$fname,$lname,$phone,$address)
{
    require ('connection.php');
```

This code checks if the database connection was successful.
If the connection fails, it displays an error message and exits the script.

```
 if (!$conn) {
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
        exit;
    }
```

This code sanitizes the input values to prevent SQL injection attacks.

It uses the mysqli_real_escape_string function to escape special characters in the input values.

It also checks if any of the input values are empty, and if so, it returns false.

```
    $userid = mysqli_real_escape_string($conn, $userid);
    $fname = mysqli_real_escape_string($conn, $fname);
    $lname = mysqli_real_escape_string($conn, $lname);
    $phone = mysqli_real_escape_string($conn, $phone);
    $address = mysqli_real_escape_string($conn, $address);
    if (empty($userid) or empty($fname) or empty($lname) or empty($phone)
or empty($address)){
    return false;
```

This code prepares and executes an SQL UPDATE query to update the user's profile information in the database.
It uses the sanitized input values to set the fname, lname, address, and phonenumber fields in the userinfo table.
If the query fails, it displays an error message and returns false.

```
    $updatequery = "UPDATE userinfo SET
fname='$fname',lname='$lname',address='$address',phonenumber='$phone'
WHERE userid='$userid'";

    $udatequery="UPDATE userinfo SET
fname='$fname',lname='$lname',address='$address',phone='$phone' WHERE
userid='$userid' IS NOT NULL;";
    if (!$queryrun = mysqli_query($conn,$updatequery)) {
        echo "Error updating record: " . mysqli_error($conn);
        return false;
    }
```

This code checks if the UPDATE query affected any rows in the database.
If at least one row was affected, it returns true.
If no rows were affected, it returns false.

```
    if (mysqli_affected_rows($conn) > 0) {
            echo "true";
        return true;
    } else {
        echo "fasle";
        return false;
    }
```

Search Trips(searchtrip.php)

This code defines a PHP function named "searchTrips" that searches for bus trips based on certain criteria. Here are the detailed comments for each part of the code:

This line defines the function "searchTrips" that takes four parameters: $busdepart (departure time), $busarrival (arrival time), $begin_locationid (starting location ID), and $end_locationid (ending location ID).

```
function searchTrips($busdepart, $busarrival, $begin_locationid,
$end_locationid)
```

This line defines the SQL query with placeholders for the search criteria. The query selects all columns from the "busschedule" table where the departure time, arrival time, starting location ID, and ending location ID match the parameters passed to the function.

```
$sql = "SELECT * FROM busschedule WHERE busdepart=? AND busarrival=? AND
begin_locationid=? AND end_locationid=?";
```

These lines prepare the SQL query and bind the parameters to the placeholders. The "mysqli_prepare" function creates a prepared statement object that represents the SQL query, and the "mysqli_stmt_bind_param" function binds the four parameters to the placeholders in the query.

```
$stmt = mysqli_prepare($conn, $sql);
mysqli_stmt_bind_param($stmt, "ssdd", $busdepart, $busarrival,
$begin_locationid, $end_locationid);
```

These lines execute the prepared statement and retrieve the results. The "mysqli_stmt_execute" function executes the prepared statement, and the "mysqli_stmt_get_result" function retrieves the result set from the prepared statement.

```
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
```

These lines check if any trips were found and display the results. The "mysqli_num_rows" function returns the number of rows in the result set, and if it is zero, the code displays the "No trips found." message. If there are rows in the result set, the code loops through each row using the "mysqli_fetch_assoc" function and displays the details of each trip.

```
if (mysqli_num_rows($result) == 0) {
    echo "No trips found.";
} else {
    // Loop through the results and display each trip
    while ($row = mysqli_fetch_assoc($result)) {
        echo "Buss ID: " . $row["busscheid"] . "<br>";
        echo "Departure time: " . $row["busdepart"] . "<br>";
        echo "Arrival time: " . $row["busarrival"] . "<br>";
        echo "Departure location: " . $row["begin_locationid"] . "<br>";
        echo "Arrival Locartion: " . $row["end_locationid"] . "<br>";
    }
}
```

Update Trip(updatetrips.php)

This is a PHP function named "updatetrip" that updates the bus schedule of a ticket in a database. Here are the comments for each part of the code:

The function is declared with two parameters: $ticketid and $busscheduleid. These are the ID of the ticket and the ID of the new bus schedule to be assigned to the ticket.

```
function updatetrip($ticketid,$busscheduleid)
```

This if statement checks if the connection to the database is successful. If not, it displays an error message and exits the script.

```
    if (!$conn) {
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
        exit;
    }
```

To prevent SQL injection attacks, these lines sanitize the input values by escaping special characters in the ticket ID and bus schedule ID.

```
   $ticketid = mysqli_real_escape_string($conn, $ticketid);
        $busscheduleid = mysqli_real_escape_string($conn, $busscheduleid);
```

This is an if statement that checks if either the ticket ID or bus schedule ID is empty. If either one is empty, the function returns false.

```
if (empty($ticketid) or empty($busscheduleid)){
    return false;
```

This line creates an SQL query to update the bus schedule ID of the ticket in the database.

```
$seltrip=" UPDATE ticket SET busscheid = $busscheduleid WHERE ticketid =
$ticketid IS NOT NULL;";
```

This line executes the SQL query and checks if it was successful. If there was an error, it displays the error message and returns false.

```
if (!$queryrun = mysqli_query($conn,$seltrip)) {
    echo "Error updating record: " . mysqli_error($conn);
    return false;
```

This if statement checks if any rows were affected by the update query. If so, it returns true;

otherwise, it returns false. The function also echoes "true" or "false" for debugging purposes.

```
 if (mysqli_affected_rows($conn) > 0) {
        echo "true";
    return true;
} else {
    echo "false";
    return false;
}
```

TESTING CODE DOCUMENTATION

Booking History Test(BookingHistoryTest.php)

This is a PHP unit test file that tests the bookinghistory function, which is expected to retrieve the booking history of a user from a database. Here is a detailed breakdown of each part of the code:

These lines include the bookinghistory function and the database connection code required for this unit test.
This line includes the TestCase class from the PHPUnit framework, the base class for all unit tests in PHPUnit.

```
require "app/bookinghistory.php";
require "app/connection.php";

use PHPUnit\Framework\TestCase;
```

This function is a test function for a valid user ID with a booking history. It checks if the expected output from the bookinghistory function with a valid user ID equals the actual output.

```
   public function testValidUserIdWithHistory() {
        // test function code here
    }
```

This function is a test function for an invalid (non-existent) user ID. It checks if the expected output from the bookinghistory function with a non-existent user ID is equal to the actual output.

```
   public function testInvalidUserIdNonExistent() {
        // test function code here
    }
```

This function is a test function for an invalid (empty string) user ID. It checks if the expected output from the bookinghistory function with an empty string user ID is equal to the actual output.

```
    public function testInvalidUserIdEmptyString() {
        // test function code here
    }
```

This line sets the user ID to 1, which is used in the first test function.

This line starts output buffering, which captures the output generated by the bookinghistory function. Then calls the bookinghistory function with the user ID. Then retrieves the output generated by the bookinghistory function and cleans the output buffer.

Then $this line asserts that the expected and actual outputs from the booking history function are equal.

```
 public function testValidUserIdWithHistory() {
        $userid = "1";
        ob_start();
        bookinghistory($userid);
        $actual_result = ob_get_clean();
        $expected_result = "Ticket ID: 1<br>first Name: victor<br>Seat
Name: School bus<br>Begin Location: Belmopan<br>Begin depart:
09:00:00<br>Cost: 0<br>";
        $this->assertEquals($expected_result, $actual_result);
    }
```

The second and third test functions are similar to the first but have different user IDs and expected outputs. The second function tests for a non-existent user ID, while the third function tests for an empty string user ID.

Check Payment Test(CheckPaymentInfoTest.php)

This first section of code includes two require statements to include the necessary code for the test to work properly. It also includes the necessary TestCase class to allow the test to run properly. The CheckPaymentInfoTest class is then created to house the test cases. The first test case method is defined and is named testValidUserId(). The expected output for this test case is defined and is a string containing the desired result from the paymentinfo function when a valid user ID is passed. The ob_start() function is called to start output buffering. The paymentinfo() function is called with a valid user ID. The ob_get_clean() function is called to retrieve the output from the paymentinfo() function and clean the output buffer.
The var_dump() function is called to print the actual result of the paymentinfo() function for debugging purposes. Finally, the assertEquals() function is called to assert that the expected output and actual output from the paymentinfo() function are equal.

```php
<?php
require "app/checkpaymentinfo.php";
require "app/connection.php";

use PHPUnit\Framework\TestCase;

class CheckPaymentInfoTest extends TestCase
{
    public function testValidUserId()
    {
        $expected_result = "Ticket ID: 1<br>first Name: victor<br>Seat
Name: School bus<br>Begin Location: Belmopan<br>Begin depart:
09:00:00<br>Cost: 0<br>";
        ob_start();
        paymentinfo(1);
        $actual_result = ob_get_clean();
        var_dump($actual_result);s
        $this->assertEquals($expected_result, $actual_result);
    }
```

The second test case method is defined and is named testInvalidUserId(). The expected output for this test case is defined and is a string containing the expected output from the paymentinfo function when passed a non-existent user ID. The ob_start() function is called to start output buffering. The paymentinfo() function is called with a non-existent user ID. The ob_get_clean() function is called to retrieve the output from the paymentinfo() function and clean the output buffer. Finally, the assertEquals() function is called to assert that the expected output and actual output from the paymentinfo() function are equal.

```
 public function testInvalidUserId()
    {
        $expected_result = "No Payment Info Found"; // expected output
        ob_start(); // starts output buffering
        paymentinfo(1000); // calls the function with a non-existent user
ID
        $actual_result = ob_get_clean(); // retrieves the output and
cleans the output buffer
        $this->assertEquals($expected_result, $actual_result); // asserts
that the expected and actual outputs are equal
    }
```

The testNullUserId function is similar to the testNoPaymentForUser function, but it tests the case where the user ID passed to the paymentinfo function is 0. This edge case should also return the "No Payment Info Found" output.

The expected output, output buffering, and assertion steps are the same as in the testNoPaymentForUser function.

```
    public function testNoPaymentForUser()
    {
        $expected_result = "No Payment Info Found"; // expected output
        ob_start(); // starts output buffering
        paymentinfo(2); // calls the function with a user ID with no
payment info
        $actual_result = ob_get_clean(); // retrieves the output and
cleans the output buffer
        $this->assertEquals($expected_result, $actual_result); // asserts
that the expected and actual outputs are equal
    }

    public function testNullUserId()
    {
        $expected_result = "No Payment Info Found"; // expected output
        ob_start(); // starts output buffering
        paymentinfo(0); // calls the function with a null user ID
        $actual_result = ob_get_clean(); // retrieves the output and
cleans the output buffer
        $this->assertEquals($expected_result, $actual_result); // asserts
that the expected and actual outputs are equal
    }
```

EditUserProfileTest(EditUserProfileTest.php)

The testUpdateUserInfoWithValidInput function is a unit test for the edithprofile function with valid input parameters. The function sets the input parameters and the expected result, calls the edithprofile function with these parameters, and asserts that the expected and actual results are equal.

```
 public function testUpdateUserInfoWithValidInput()
    {
        $userid = 1;
        $fname = 'Adaasddsfsfaadsdasdsd';
        $lname = 'Adasdssadsfsdasddadasaadad';
        $phone = '4443333844';
        $address = '12sas333add3 Mawdadain St.';
        $expected_result = true;

        $actual_result = edithprofile($userid, $fname, $lname, $phone,
$address);

        $this->assertEquals($expected_result, $actual_result);
    }
```

The testUpdateUserInfoWithEmptyFirstName function is a unit test for the edithprofile function with an empty first name. The function sets the input parameters and the expected result, calls the edithprofile function with these parameters, and asserts that the expected and actual results are equal.

```
  public function testUpdateUserInfoWithEmptyFirstName()
    {
        $userid = 1;
        $fname = '';
        $lname = 'Doe';
        $phone = '1234567890';
        $address = '123 Main St.';
        $expected_result = false;

        $actual_result = edithprofile($userid, $fname, $lname, $phone,
$address);

        $this->assertEquals($expected_result, $actual_result);
    }
```

In this function testUpdateUserInfoWithInvalidUserId, an invalid user ID is set as a string, which would not be accepted as a valid user ID in the application's database. The function then sets other valid input values for the user's first name, last name, phone number, and address. The expected result for this test case is false because the user ID is invalid, and the update should not be successful. The edithprofile function is then called with the input parameters, and the actual result is stored in the $actual_result variable.

```
  public function testUpdateUserInfoWithInvalidUserId()
    {
        $userid = 'invalid';
        $fname = 'John';
        $lname = 'Doe';
        $phone = '1234567890';
        $address = '123 Main St.';
        $expected_result = false;

        $actual_result = edithprofile($userid, $fname, $lname, $phone,
$address);

        $this->assertEquals($expected_result, $actual_result);
    }
```

In this function testUpdateUserInfoWithInvalidPhoneNumber, the input parameter for the user ID is set to null, indicating that the function should not update any user's information. The phone number is also set to an invalid value, which would not be accepted as a valid phone number in the application's database. The expected result for this test case is false because the phone

number is invalid and the update should not be successful. The edithprofile function is then called with the input parameters and the actual result is stored in the $actual_result variable.

```
   public function testUpdateUserInfoWithInvalidPhoneNumber()
    {
        $userid = null; /
        $fname = 'John';
        $lname = 'Doe';
        $phone = 'invalid';
        $address = '123 Main St.';
        $expected_result = false;

        $actual_result = edithprofile($userid, $fname, $lname, $phone,
$address);

        $this->assertEquals($expected_result, $actual_result);
```

SearchTripsTest(SearchTripsTest.php)

The code defines a PHP unit test class called SearchTripsTest, extending the TestCase class from PHPUnit. The purpose of the test class is to test the searchTrips function defined in the searchtrip.php file.

The test class contains five test methods that test different scenarios of the search tips function:

The testValidSearch(): This method tests the searchTrips function when valid parameters are passed. This method starts an output buffering by calling the ob_start function, then it calls the searchTrips function with correct parameters, and captures the result of the function in the $search_result variable by calling the ob_get_clean function. Then, it defines the expected output of the function in the $expected_output variable, and uses PHPUnit's assertStringContainsString method to compare the desired result with the actual production.

```
public function testValidSearch()
    {
        ob_start();
        searchTrips("08:00:00", "08:55:00", 1, 2);
        $search_result = ob_get_clean();
```

```
        var_dump($search_result);
        $expected_output = "Buss ID: 1<br>Departure time:
08:00:00<br>Arrival time: 08:55:00<br>Departure location: 1<br>Arrival
Locartion: 2<br>";
$this->assertStringContainsString($expected_output, $search_result);
    }
```

The testNoTripsFound(): This method tests the searchTrips function when no trips are found for the specified parameters. The method starts an output buffering, calls the searchTrips function with invalid parameters, captures the output of the function in the $search_result variable, and uses PHPUnit's assertEquals method to compare the expected output with the actual output.

```
public function testNoTripsFound()      {
        ob_start();
        searchTrips("09:00:00", "11:00:00", 2, 3);
        $search_result = ob_get_clean();
        $this->assertEquals('No trips found.', $search_result); \
    }
```

The testInvalidLocationIds(): This method tests the searchTrips function when invalid location IDs are passed as parameters. The method starts an output buffering, calls the searchTrips function with invalid parameters, captures the output of the function in the $search_result variable, and uses PHPUnit's assertEquals method to compare the expected output with the actual output.

```
    public function testInvalidLocationIds()
    {
        ob_start();
        searchTrips("09:00:00", "11:00:00", "invalid", "location");
        $search_result = ob_get_clean();
        $this->assertEquals('No trips found.', $search_result);
    }
```

The testInvalidTimeFormat(): This method tests the searchTrips function when invalid time formats are passed as parameters. The method starts an output buffering, calls the searchTrips function with invalid parameters, captures the function output in the $search_result variable, and uses PHPUnit's assertEquals method to compare the expected output with the actual output.

```
public function testInvalidTimeFormat()t
 {
     ob_start();
     searchTrips("9:00", "11:00", 1, 2);
     $search_result = ob_get_clean();
     $this->assertEquals('No trips found.', $search_result);
 }
```

The testEmptyParameters(): This method tests the searchTrips function when empty parameters are passed. The method starts an output buffering, calls the searchTrips function with empty parameters, captures the function output in the $search_result variable, and uses PHPUnit's assertEquals method to compare the expected output with the actual output.

```
public function testEmptyParameters()
 {
     ob_start();
     searchTrips("", "", "", "");
     $search_result = ob_get_clean();
     $this->assertEquals('No trips found.', $search_result);
 }
```

Update Trip Test(UpdateTripTest.php)

The testValidInput() method tests the updatetrip function with valid input parameters. It sets the values of $ticketid and $busscheduleid to 1, calls the updatetrip function with these parameters, captures the output with the "var_dump" function, sets the expected result to true, and compares the expected and actual results using the assertEquals method.

```
public function testValidInput()
{
    $ticketid = 1;
    $busscheduleid = 1;
    $actual_result = updatetrip($ticketid, $busscheduleid);
    var_dump($actual_result);
    $expected_result = true;

    $this->assertEquals($expected_result, $actual_result);
}
```

The testInvalidBusScheduleId() method tests the updatetrip function with an invalid $busscheduleid parameter. It sets the value of $ticketid to 1 and $busscheduleid to an empty string, calls the updatetrip function with these parameters, captures the output with the var_dump function, sets the expected result to false, and compares the expected and actual results using the "assertEquals" method.

```
public function testInvalidBusScheduleId()
{
    $ticketid = 1;
    $busscheduleid = '';
    $actual_result = updatetrip($ticketid, $busscheduleid);
    var_dump($actual_result);
    $expected_result = false;

    $this->assertEquals($expected_result, $actual_result);
}
```

The testInvalidId() method tests the updatetrip function with an invalid $ticketid parameter. It sets the value of $ticketid to an empty string and $busscheduleid to 5, calls the updatetrip function with these parameters, captures the output with the var_dump function, sets the expected result to false, and compares the expected and actual results using the assertEquals method.

```
public function testInvalidId()
{
    $ticketid = '';
    $busscheduleid = 5;
    $actual_result = updatetrip($ticketid, $busscheduleid);
    var_dump($actual_result);
    $expected_result = false;

    $this->assertEquals($expected_result, $actual_result);
}
```