

Question 1: MVC Responsibility Identification

a) Calculating the total price including tax and discounts

Component: Model

Explanation:

Price calculation, tax addition, and discount application are part of business logic. Such logic should be placed in the Model because the Model is responsible for data processing and application rules.

b) Rendering the shopping cart page with product thumbnails

Component: View

Explanation:

The View is responsible for presenting data to the user. Displaying product names, prices, and thumbnails is a visual task, so it belongs to the View.

c) Processing the “Add to Cart” button click

Component: Controller

Explanation:

The Controller handles user interactions. When the user clicks the “Add to Cart” button, the Controller receives the request and decides how to process it.

d) Querying the database for products matching search criteria

Component: Model

Explanation:

Database access and queries are handled by the Model because it manages data storage and retrieval.

e) Validating that quantity is a positive integer before updating cart

Component: Controller

Explanation:

Input validation is performed by the Controller before sending data to the Model. This ensures that only valid input is processed.

Question 2: MVC Mini Application – Student Grade System

Model (StudentModel.php)

```
<?php  
  
class StudentModel {  
  
    public function getStudents() {  
        // Fetch students from database  
    }  
  
    public function addGrade($studentId, $grade) {  
        // Insert grade into database  
    }  
  
    public function calculateAverage() {  
        // Calculate class average  
    }  
}  
?>
```

Explanation:

The Model handles all database operations and grade calculations.

Controller (GradeController.php)

```
<?php  
include "StudentModel.php";  
  
$model = new StudentModel();  
  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    $studentId = $_POST["student_id"];  
    $grade = $_POST["grade"];  
  
    if ($grade >= 0 && $grade <= 100) {  
        $model->addGrade($studentId, $grade);  
    }  
}  
?>
```

Explanation:

The Controller receives form input, validates it, and calls the appropriate Model methods.

View

The View displays:

- Student list
- Grade input form
- Class average

The View does not contain database logic.

Question 3: Separation of Concerns

Separation of concerns means dividing an application into parts where each part has a specific responsibility.

Placing database queries inside the View is bad practice because:

- It mixes logic with presentation
- Makes code hard to maintain
- Violates MVC principles
- Increases duplication and errors

Database logic should always be placed in the Model.

Question 4: Real-Time Comment Preview Using AJAX

AJAX allows data to be sent to the server without reloading the page.

HTML & JavaScript

```
<!DOCTYPE html>

<html>
<body>

<textarea id="comment" onkeyup="preview()"></textarea>
<div id="result"></div>

<script>
function preview() {
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "preview.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            document.getElementById("result").innerHTML = xhr.responseText;
        }
    }
}
</script>
```

```
        }
    };

    xhr.send("comment=" + document.getElementById("comment").value);
}

</script>
```

```
</body>
```

```
</html>
```

Explanation:

User input is sent to the server in the background, and the preview is shown without page reload.

Question 5: AJAX Bug Analysis

a) Bug Explanation

`response.message` is incorrect because the response is a string, not an object.

b) Missing Content-Type

Without setting content type, PHP may not correctly read POST data.

c) Error Handling

```
xhr.onerror = function() {
    document.getElementById("result").innerHTML = "Error occurred";
};
```

d) readyState Values

- 0 – Request not initialized
- 1 – Server connection established
- 2 – Request received
- 3 – Processing request

Question 6: AJAX vs Traditional Form Submission

Feature	AJAX	Traditional
---------	------	-------------

Page reload	No	Yes
-------------	----	-----

Speed	Faster	Slower
-------	--------	--------

User experience	Better	Basic
-----------------	--------	-------

Complexity	Higher	Lower
------------	--------	-------

Traditional forms are better for simple submissions and file uploads.

Question 7: Shopping Cart Using Session

```
<?php  
session_start();  
  
if (!isset($_SESSION["cart"])) {  
    $_SESSION["cart"] = [];  
}
```

```
function addItem($name, $price, $qty) {  
    $_SESSION["cart"][$name] = [  
        "price" => $price,  
        "quantity" => $qty  
    ];  
}
```

```
function removeItem($name) {
```

```

unset($_SESSION["cart"][$name]);

}

function getTotal() {
    $total = 0;
    foreach ($_SESSION["cart"] as $item) {
        $total += $item["price"] * $item["quantity"];
    }
    return $total;
}
?>

```

Explanation:

Sessions store cart data on the server and persist across pages.

Question 8: Security Issues

a) Session hijacking

Stealing a valid session ID.

b) HttpOnly flag

Prevents JavaScript from accessing cookies.

c) Storing passwords in cookies

Unsafe because cookies can be stolen.

d) Secure Cookie Example

```
setcookie("user","admin",time()+3600,"/", "", true, true);
```

Question 9: Session & Cookie Tracing

- First visit → count = 1

- Second visit → count = 2
 - Browser closed → session reset
 - Cookie expiry → new visit count
-

Question 10: Output Prediction

- Snippet 1 → A B O D
 - Snippet 2 → 4 cherry
 - Snippet 3 → 11 vs 13
 - Snippet 4 → fg cdef
-

Question 11: include vs require

Function Behavior

include Warning on failure

require Fatal error

*_once Included once only

Question 12: Input Validation Function

```
function validateData($data) {  
    $errors = [];  
  
    if (empty($data["name"])) {  
        $errors[] = "Name required";  
    }  
  
    if (!filter_var($data["email"], FILTER_VALIDATE_EMAIL)) {
```

```
$errors[] = "Invalid email";  
}  
  
if ($data["age"] < 13) {  
    $errors[] = "Invalid age";  
}  
  
return $errors;  
}
```

Question 13: MySQLi vs PDO

Feature	MySQLi	PDO
Database support	MySQL only	Multiple
Error handling	Manual	Exception based
Prepared statements	Yes	Yes

Question 14: SQL Injection Prevention

Prepared statements prevent SQL injection by separating SQL logic from user input.

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username=?");  
$stmt->bind_param("s", $username);  
$stmt->execute();
```