Inventory Management System Report

**Date:** 16 / 1 / 2024

**Executive Summary:**

In this part we are going to give a brief explanation of what our website is, our website is used to facilitate the work of managers, from sales, products, orders, suppliers, users, to statistical survey.  
  
 The objectives of this website is to present the manager’s responsibilities in a professional way, in which everything is presented in tables , forms and graphs thus it is easy on him to print every piece of information and evidence.

Moreover, every detail is dated thus the history of the order, adding a supplier, receiving orders, and updating data is all saved for more precision, safety, and accuracy.

Of course our website is based on main keys such as:

* Product Management.
* Supplier Tracking.
* Orders Management.
* Users access control.
* Statistical dashboard.

All these details mentioned above will improve the efficiency, reduce errors, better decision making and overview, real-time visibility, scalability, enhanced accuracy and time savings.

**Website Detailed Explanation :**

1. **Home Page:**

The homepage is the welcoming page of our website, it contains our website’s logo which is IMS that stands for “Inventory Management System”.

The background we chose for our site’s homepage is this one, because as you can see it contains shelves that are filled with products in a supermarket which matches our websites needs and theme, when you scroll down the picture stays fixed and the other elements that will be mentioned below will appear above it.

It also contains links for changing the theme in order to customize the look and make the user feel comfortable with it, flat design in order to enhance the user’s experience and keeps everything simple, and reach your audience to connect with other managers globally and manage the inventory with ease. When you hover above them the container they belong to becomes somehow how opaque in which the transparency increases.  
  
Moreover, there is an option to enter your email address in order to get notified for any update we have created or any important announcements about our inventory management system. Next to it comes a YouTube video associated to it for more explanation and clearance .  
  
Furthermore, comes a group of links to keep in touch with us on social media, because nowadays social media is important and so active.  
These links are for twitter, face book, pintrest, Google, LinkedIn, and finally YouTube.  
At the bottom of the homepage we have a section that contains the following options: contact, download, press, Email, support, and finally the Privacy Policy. If you hover above any one of them it becomes underlined.

Basically the main color we are using in the homepage as well as the other pages is this color because it is pleasant to look at and the contrast with black matches as well as white.  
  
The most important part of our homepage is the top navigation bar that is always fixed while scrolling, it contains the login that should be pressed in order for the user to be able to benefit from our website and go to the main part.

1. **Login Page:**

The login page appears after pressing the login button in the homepage.  
It contains the title “Inventory Management System” and a form in order for the user to enter his email and password.  
If the user presses the login button with the textfields being empty a message stating that “You have to fill both the email and password !” appears.

Of course the email and password of each user is entered in the database and specified, thus not any email or password will work, if the email was entered correctly however its password was incorrect for three times or more the account will be blocked else if the login went on successfully the user will be able to be transformed to the main page which is the page that contains all the data and information, the dashboard page.

Take into consideration that a message stating that there was an error in the email or password will be presented if one of them was entered incorrectly.

1. **Dashboard page:**

The dashboard page contains the information related to the product countassigned to supplier in a bar graph, in which the bars are colored in different colors to enable the manager to be visually focused and see which supplier has the best sales accordingly.  
  
Plus, is the pie chart that shows the information related to the purchase orders by status whether they are:

-complete

- pending

- Incomplete

You will be able to transform the present data into a pdf document, or save it as an image with different types like PNG, JPEG, SVG, or print the charts immediately.

To add you can view the charts as a full page to see everything in a more clear way and press an option associated to the analysis of these charts by giving the breakdown of whatever graph you chose.

Concerning the design, we have a very important note here:

SIDEBAR NOTE !!!

This page is formed of a sidebar that is fixed in all of the pages, this sidebar contains a list of the options we want to redirect us to the page preferred the options found in it are the following:

* Dashboard
* Reports
* Product
* View Products
* ADD Products
* Supplier
* View Suppliers
* ADD Suppliers
* Purchase Order
* View Orders
* ADD Orders
* User
* View Users
* ADD Users

The options such as { Product / Supplier / Purchase Order / User } if pressed in the side bar present,the triangular arrow next to them will be faced from the initial state which is to the left towards the new state which is downward and the sub options each one has the { View / ADD} will appear. If pressed again the options will hide and the arrow will return back to its initial state.  
  
The sidebar is fixed in all of the 14 pages found in this website, so that you can jump from a page to another immediately, however the content of the page will of course change according to the option chosen in the sidebar.

NAVIGATION BAR NOTE !!!

Like the sidebar that is present in all of the 14 pages we have the navigation bar fixed too, the navigation bar contains an essential button which is the logout button, this button enables the user to return back to the login page and login again.  
  
After the user logs in if he presses the back arrow found on the top left of the page he won’t be able to return to the login page, because he already logged in, thus the logout button has the job of returning the user to the login page and removing his previous submission.

Now the user has to enter his email and password again to be able to work in the main pages later.

1. **User => View Users**

The view users page contains the fixed sidebar and navigation bar previously described, but it also contains a table that presents the list of users mainly managers that can access the website and login successfully.

The information presented in this table are the following:

* The number of the user
* First name
* Last name
* Email
* Created at
* Updated at
* Action

The first four checks are clear and trivial however concerning the {Created at } this date shows when did this specified user be added to the system to have the ability to access it.

The { Updated at } shows the date the information of this user were updated for example his email.

The { Action } column contains for each row two links, the first one is the edit link and the second one is the delete link.

* From their names the { edit } link is used for editing the data of the specified user, when it is clicked a new form will appear having inputs to be filled, if they are filled successfully a message stating that the update was successful will appear.
* The { delete } link is used to delete this user from the system and prevents him from logging in again unless his data is added to the system again, before deleting a question will appear asking if we are sure that we want to delete this user, if we press ok then this deletion will occur and a message saying that the deletion was successful will appear.

Finally, the number of users is also presented.

1. **Users => ADD User :**

This page contains the fixed sidebar and navigation bar.  
  
The content of it is mainlypresented in a form, this form contains:

* First name with a textfield
* Last name with a textfield
* Email with a textfield
* Password with a textfield
* Add user button

Everything the manager has to do is fill in the form and submit the data to add a new user using the add user button.

If he pressed the button with empty text fields an error message will be presented and he will be notified to fill in all of the blanks.

1. **Supplier => View Suppliers:**

This pages contains the fixed side and navigation bars.

It presents the list of suppliers we are working with in a table, these are the following columns in the table:

* Number of supplier
* Supplier name
* Supplier location
* Contact details (email)
* Products
* Created by
* Created at
* Updated at
* Action

The { products } attribute shows the products that we entered to the system and that belong to the specified supplier.

The {created by } attribute specifies which user added this supplier to the list of suppliers .

The { created at / updated at / action } are the same as the ones previously explained .

Finally, the number of suppliers is also presented.

1. **Supplier = > ADD Suppliers :**

This page contains the form that enables the manager to add a new supplier to the system, this form contains:

* Supplier name with a textfield
* Location with a textfield
* Email with a textfield

The user should fill all of them before submitting the form to successfully add this supplier and have a success message presented; otherwise an error message will appear.

1. **Product => View Products :**

This page contains the list of products presented in a table, here are the columns of it:

* Number of product
* Image
* Product name
* Description
* Suppliers
* Created by
* Created at
* Updated at
* Action

The { description } describes this product .

The { Image } gives a small image associated to each product to facilitate its significance.

The { Suppliers } shows who are the suppliers relative to this product.

The ( created by / created at / updated at / Action } are explained before.

And the number of products is presented.

1. **Products => ADD products:**

The add products page contains a form that has the following data to be filled:

* Product name with a textfield
* Description with a textfield
* Supplier with a select and option of suppliers entered to the system
* Product image with a choose button to enable the user to choose a photo from a specified file.
* Create Product button to submit the form data.

If the form was submitted without choosing a supplier, entering the name of the product and choosing a photo an error message will appear otherwise a message stating that the product was successfully added to the system will appear.

1. **Purchase Orders => View Orders:**

This page contains the list of orders that are found in the system, you will see on top of each order a batch number, this number is a random number used to differentiate between all the orders.  
  
Each order will have the following information:

* The number of order
* Product ordered
* Quantity ordered
* Quantity received
* Supplier
* Status ( incomplete / complete / pending )
* Ordered by
* Created date
* Delivery History (deliveries button)
* Remove Order button
* Update button

If the deliveries button is pressed it will show a table that pops up presenting the quantity received and in what date

.The remove button asks the user if he is sure that he wants to delete this option, if yes he deletes this order and presents a message saying that the deletion was done successfully.

The update button updates the order, from its quantity received and it status.

1. **Purchase Order => ADD Order :**

This page enables the user from adding another order, it contains a wide Add another Product button that is when pressed presenting a form containing:

* Product name and a select with options
* Supplier that shows the supplier of the chosen product before
* Quantity with a textfield to enter the quantity.
* Submit button.

If the user doesn’t fill in the form entirely and he presses the submit button an error will occur else a message stating that the addition of the order was successful will be presented.

1. **Reports:**

This page contains three containers these containers are the:

* Export Products
* Export Suppliers
* Export Purchase Orders

Each one of them has two buttons the excel button and the pdf button.

The excel button enables the user to present the data in excel and the pdf enables him to transform the data into a pdf document.

**Website Mechanism :**

All the details in this website are related to each other, in which if a specified supplier was deleted all the products that are owned by this supplier will be deleted too, consequently if there are any orders with this supplier they will also be deleted.

If a specified product was deleted, any order using this product will be deleted too, however if an order was deleted this doesn’t mean that the product used in it will be deleted too.

So as a conclusion there is a very tight relation between the:

Supplier

Order Product

Were the supplier is the big boss followed by the product because if a specific supplier is removed from the system everything related to it in the system will be removed too.

**DATABASE :**

Our database’s name was “inventory”, it contained 5 tables that have the following names:

* Products
* Suppliers
* Productsuppliers
* Users
* Order\_product

The Products table contains the following attributes:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | Product\_name | description | img | Created\_by | Created\_at | Updated\_at |

* ( Id ) is the primary key.
* ( created\_by ) is a foreign key that has a relation with the table Users with its id.

The Suppliers table contains the following attributes:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | Supplier\_name | Supplier\_location | email | Created\_by | Created\_at | Updated\_at |

* ( id ) is the primary key.
* ( created\_by ) is a foreign key in a relation with the Users table specifically its id.

The Productsuppliers table contains the following attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | supplier | product | Updated\_at | Created\_at |

* (id) is the primary key.
* ( supplier ) is a foreign key with the Suppliers table specifically its id and on delete it is set to cascade.
* ( product ) is a foreign key with Products table specifically its id and on delete it is set to cascade.

Cascade is used to delete the product in the products table if the supplier is deleted from the suppliers table and it is set in the productsuppliers table because it is the table that presents the relation between the products and the suppliers.

The Users table contains the following attributes:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | First\_name | Last\_name | password | email | Created\_at | Updated\_at | status |

* ( id ) is the primary key here and we don’t have foreign keys.

The Order\_product table contains the following attributes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | supplier | product | Quantity\_ordered | Quantity\_received | Quantity\_remaining |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| status | batch | Created\_by | Created-at | Updated\_at | Received\_at |

* ( id ) is the primary key
* ( supplier ) is a foreign key with the Suppliers table specifically its id and its delete is set to cascade.
* ( product ) is a foreign key with the Products table specifically its id  
  and its delete is set to cascade.
* ( created\_by ) is a foreign key with the Users table specifically its id.

**SQL Note Related to our website’s mechanism:**

We used this Query on the Suppliers table in order for the deletion of the products related to it to occur :

DELIMETER / /

CREATE TRIGGER after\_delete\_supplier

AFTER DELETE ON suppliers FOR EACH ROW

BEGIN

DECLARE supplier\_id INT;

SET supplier\_id=OLD.id;

DELETE FROM productsuppliers WHERE supplier = supplier\_id;

DELETE FROM products WHERE id NOT IN (SELECT

DISTINCT product FROM productsuppliers);

END ;

/ /

DELIMITER ;

This is the most important note concerning our database sql query.

**CODING PART :**

In order to form our website we used several programming languages such as :

* Html
* Css
* Javascript
* Php
* Ajax
* Bootstrap
* JQuery

The new used languages will have courses that explain what are they and how did we benefit from them:

Starting with AJAX:

AJAX is a developer's dream, because you can:

* Update a web page without reloading the page
* Request data from a server - after the page has loaded
* Receive data from a server - after the page has loaded
* Send data to a server - in the background

What is AJAX?

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

* A browser built-in XMLHttpRequest object (to request data from a web server)
* JavaScript and HTML DOM (to display or use the data)
* AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

AJAX Example Explained:

<!DOCTYPE html>  
 <body>  
  
 <div id="demo">  
   <h2>Let AJAX change this text</h2>  
   <button type="button" onclick="loadDoc()">Change Content</button>  
 </div>  
  
 </body>  
 </html>

The HTML page contains a <div> section and a <button>.

The <div> section is used to display information from a server.

The <button> calls a function (if it is clicked).

The function requests data from a web server and displays it:

function loadDoc() {  
  var xhttp = new XMLHttpRequest();  
  xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
     document.getElementById("demo").innerHTML = this.responseText;  
    }  
  };  
  xhttp.open("GET", "ajax\_info.txt", true);  
  xhttp.send();  
}

The "ajax\_info.txt" file used in the example above, is a simple text file and looks like this:

<h1>AJAX</h1>  
<p>AJAX is not a programming language.</p>  
<p>AJAX is a technique for accessing web servers from a webpage.</p>  
<p>AJAX stands for Asynchronous JavaScript And XML.</p>

Now every part of the previous code example will be explained in details:

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, Edge (and IE7+), Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

### Example

var xhttp = new XMLHttpRequest();

XMLHttpRequest Object Methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| new XMLHttpRequest() | Creates a new XMLHttpRequest object |
| abort() | Cancels the current request |
| getAllResponseHeaders() | Returns header information |
| getResponseHeader() | Returns specific header information |
| open(*method,url,async,user,psw*) | Specifies the request  *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password |
| send() | Sends the request to the server Used for GET requests |
| send(*string*) | Sends the request to the server. Used for POST requests |
| setRequestHeader() | Adds a label/value pair to the header to be sent |

## XMLHttpRequest Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| onreadystatechange | Defines a function to be called when the readyState property changes |
| readyState | Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready |
| responseText | Returns the response data as a string |
| responseXML | Returns the response data as XML data |
| status | Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the [Http Messages Reference](https://www.w3schools.com/tags/ref_httpmessages.asp) |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

xhttp.open("GET", "ajax\_info.txt", true);  
 xhttp.send();

GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

* A cached file is not an option (update a file or database on the server).
* Sending a large amount of data to the server (POST has no size limitations).
* Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

# AJAX - Server Response

## The onreadystatechange Property

The **readyState** property holds the status of the XMLHttpRequest.

The **onreadystatechange** property defines a function to be executed when the readyState changes.

The **status** property and the **statusText** property holds the status of the XMLHttpRequest object.

The onreadystatechange event is triggered four times (1-4), one time for each change in the readyState.

The onreadystatechange function is called every time the readyState changes.

When readyState is 4 and status is 200, the response is ready:

function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("demo").innerHTML =  
            this.responseText;  
       }  
    };  
    xhttp.open("GET", "ajax\_info.txt", true);  
    xhttp.send();  
}

Server Response Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| responseText | get the response data as a string |
| responseXML | get the response data as XML data |
|  |  |

Now that you have an idea of what AJAX is we can talk about JQuery which is mainly used in our code .

**Lets talk about JQuery;**

Query is a fast, small, and feature-rich JavaScript library. It simplifies many common tasks in web development, such as HTML document traversal and manipulation, event handling, and animation. Let's go through jQuery from A to Z with examples:

### 1- Adding jQuery to Your Project

To use jQuery, you need to include it in your project. You can either download it and host it locally or include it from a CDN (Content Delivery Network). Here's an example using a CDN:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>jQuery Example</title>

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

</head>

<body>

<!-- Your HTML content goes here -->

<script>

// Your jQuery code goes here

</script>

</body>

</html>

And actually this is what we did in our code :

### 2 - Basics of Selectors

One of the main features of jQuery is its powerful selector engine, allowing you to select and manipulate HTML elements easily. Here are some examples:

// Select by tag name

$('p')

// Select by class

$('.myClass')

// Select by ID

$('#myId')

// Select by attribute

$('input[name="username"]')

### 3 - CSS Manipulation

jQuery simplifies CSS manipulation. Here are some examples:

// Change text color

$('.myClass').css('color', 'red');

// Add a CSS class

$('#myId').addClass('highlight');

// Toggle a class

$('.toggleClass').toggleClass('active');

### 4 - DOM Manipulation

jQuery allows easy manipulation of the Document Object Model (DOM). Examples:

// Append content

$('#myDiv').append('<p>New paragraph</p>');

// Remove an element

$('.removeMe').remove();

// Get or set text content

var text = $('#myElement').text();

$('#myElement').text('New text');

The DOM, or Document Object Model, is a programming interface for web documents. It represents the structure of a document as a tree-like data structure where each node corresponds to a part of the document, such as an element, attribute, or piece of text. The DOM provides a way for programs (typically scripts written in languages like JavaScript) to dynamically access, manipulate, and update the content, structure, and style of a document.

### 5 - Event Handling

Event handling is simplified in jQuery. Examples:

// Click event

$('#myButton').click(function() {

alert('Button clicked!');

});

// Hover event

$('.hoverMe').hover(

function() {

$(this).css('color', 'blue');

},

function() {

$(this).css('color', 'black');

}

);

### 6 - AJAX (Asynchronous JavaScript and XML)

And this is mainly what we used in our codes, AJAX that is previously explained but here specifically as a request.

jQuery simplifies AJAX requests. Example:

$.ajax({

url: 'example.php',

method: 'GET',

dataType: 'json',

success: function(data) {

console.log('Data received:', data);

},

error: function(xhr, status, error) {

console.error('AJAX Error:', status, error);

}

});

### 7- Handling Utilities

jQuery provides various utility functions. Examples:

// Check if an element has a class

if ($('#myElement').hasClass('myClass')) {

// Do something

}

// Iterate over elements

$('li').each(function(index, element) {

console.log('Index:', index, 'Element:', element);

});

And here we will provide a part of our code that uses ajax and jquery :

$(document).on('click', '.removepoBtn', **function** (*e*) {

        e.preventDefault();

**var**orderId**=**$(this).data('id');

        console.log(orderId);

**if** (confirm('Are you sure you want to delete this order' )) {

            $.ajax({

                type: 'POST',

                data: { order\_id: orderId },

                url: 'delete-order.php',

                dataType: 'json',

                success: **function** (*data*) {

**if** (data.success) {

                        alert(data.message);

                        location.reload();

                    } **else** {

                        alert(data.message);

                    }

                },

                error: **function** (*xhr*, *status*, *error*) {

                    console.error('AJAX Error:', status, error);

                }

            });

        } **else** {

            console.log('Deletion canceled.');

        }

    });

And here is the explanation of this code that is repeated in our codes for deleting and editing mainly :

1. **$(document).on('click', '.removepoBtn', function (e) {...}**: This line attaches a click event handler to the document but filters it to only respond to clicks on elements with the class 'removepoBtn'. The function inside the **on** method is executed when an element with this class is clicked.
2. **e.preventDefault();**: Prevents the default behavior of the click event, which is to navigate to a new page.
3. **var orderId = $(this).data('id');**: Retrieves the value of the 'data-id' attribute from the clicked element using jQuery's **data** method.
4. **console.log(orderId);**: Outputs the orderId to the console for debugging purposes.
5. **if (confirm('Are you sure you want to delete this order')) {...}**: Displays a confirmation dialog. If the user clicks 'OK', the block of code inside the if statement is executed.
6. **$.ajax({ ... });**: Initiates an AJAX request to the server for deleting the order.
   * **type: 'POST'**: Specifies that it's a POST request.
   * **data: { order\_id: orderId }**: Sends the order ID to the server as part of the request data.
   * **url: 'delete-order.php'**: Specifies the URL where the request is sent.
   * **dataType: 'json'**: Expects a JSON response from the server.
7. The **success** and **error** callback functions handle the server's response. If the server indicates success (**data.success** is true), it displays a success message and reloads the page. If there's an error, it logs an error message to the console.
8. **console.log('Deletion canceled.');**: Outputs a message to the console if the deletion is canceled (if the user clicked 'Cancel' in the confirmation dialog).

**Concerning Bootstrap :**

### Definition:

**Bootstrap** is a popular open-source front-end framework developed by Twitter. It consists of a collection of HTML, CSS, and JavaScript components, providing a standardized design and layout for web pages. Bootstrap is designed to be responsive, making it easy to create websites that look good on various devices and screen sizes.

### When to Use Bootstrap and Its Purpose:

1. **Rapid Prototyping:**
   * Bootstrap is widely used for rapid prototyping and development. It allows developers to quickly create functional and visually appealing prototypes.
2. **Consistent Design:**
   * Bootstrap provides a consistent and professional-looking design across different browsers and devices. It helps maintain a cohesive appearance for web applications.
3. **Responsive Design:**
   * Bootstrap is built with a mobile-first approach, making it easy to create responsive web pages that adapt to different screen sizes, from large desktops to small mobile devices.
4. **Cross-Browser Compatibility:**
   * Bootstrap ensures compatibility across various browsers, reducing the need for extensive testing and adjustments for different platforms.
5. **Extensibility:**
   * Bootstrap is highly extensible, allowing developers to customize and extend its components based on project requirements.
6. **Community Support:**
   * Bootstrap has a large and active community, providing support, documentation, and a wide range of third-party themes and plugins.

### Using Bootstrap in JavaScript for Dialog Representation:

One of the common use cases for JavaScript in Bootstrap is creating dialog boxes or modals. Modals are overlay components that can display additional content or require user input. Bootstrap provides a robust set of modal components that can be easily customized.

Here's a simple example of creating a Bootstrap modal using JavaScript:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Bootstrap Example</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container">

<div class="row">

<class="col-sm-6">

<h2>Column 1</h2>

<p>This is the left column.</p>

</div>

<div class="col-sm-6">

<h2>Column 2</h2>

<p>This is the right column.</p>

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

We will present the bootstrap used in our code and explain this code in details , to show how simple Bootstrap makes our work :

**function**showDeliveryForm(*batchNumber*) {

*// Fetch data from the server using AJAX*

    $.ajax({

        type: 'POST',

        url: 'fetch-order-details.php',

        data: { batchNumber: batchNumber },

        dataType: 'json',

        success: **function** (*data*) {

**if** (data.success) {

*// Dynamically create the form with default values*

**var**formHtml**=**'<form id="deliveryForm"

action="process-deliveries.php" method="post">'**+**

                    '<h2>Delivery Information for Batch #: '**+**

batchNumber**+**'</h2>'**+**

                    '<table>'**+**

                    '<thead>'**+**

                    '<tr>'**+**

                    '<th>Date Received</th>'**+**

                    '<th>Quantity</th>'**+**

                    '</tr>'**+**

                    '</thead>'**+**

                    '<tbody>'**+**

                    '<tr>'**+**

                    '<td><input type="text" name="date\_received[]"

value="'**+**data.receivedDate**+**'"

required readonly></td>'**+**

                    '<td><input type="number" name="quantity[]"

value="'**+**data.quantity**+**'"

required readonly></td>'**+**

                    '</tr>'**+**

                    '</tbody>'**+**

                    '</table>'**+**

                    '</form>';

*// Show the form using BootstrapDialog*

                BootstrapDialog.show({

                    title: 'Delivery Form',

                    message: formHtml,

                    closable: true,

                    draggable: true,

                    buttons: [{

                        label: 'Close',

                        action: **function** (*dialogRef*) {

                            dialogRef.close();

                        }

                    }]

                });

            } **else** {

                alert('Failed to fetch order details. Please try

again.');

            }

          },

        error: **function** (*xhr*, *status*, *error*) {

            console.error('AJAX Error:', status, error);

        }

    });

}

1. **function showDeliveryForm(batchNumber)**:This line declares a JavaScript function named **showDeliveryForm** that takes a **batchNumber** as a parameter.
2. **$.ajax({ ... })**: This is the jQuery AJAX function that makes an asynchronous HTTP request to the server. It takes an object with various properties defining the details of the request.
   * **type: 'POST'**: Specifies that it's a POST request.
   * **url: 'fetch-order-details.php'**: Specifies the URL to which the request is sent.
   * **data: { batchNumber: batchNumber }**: Data to be sent to the server. In this case, it's the **batchNumber** parameter.
   * **dataType: 'json'**: Specifies that the expected response from the server is in JSON format.
3. **success:function (data) { ... }**: This is a callback function that will be executed if the AJAX request is successful. It receives the response from the server, and in this case, it is named **data**.
   * **if (data.success) { ... }**: Checks if the response indicates success. If **success** is true, it proceeds with creating and showing the delivery form; otherwise, it shows an alert about the failure.
   * **var formHtml = '...'**: Dynamically creates an HTML string that represents a form. It includes information from the response (**data.receivedDate** and **data.quantity**) in the default values of form fields.
   * **BootstrapDialog.show({ ... })**: Uses the BootstrapDialog library to display a modal dialog with the dynamically created form.
     + **title: 'Delivery Form'**: Sets the title of the modal.
     + **message: formHtml**: Sets the content of the modal to the dynamically created form.
     + **closable: true**: Allows the user to close the modal by clicking outside of it.
     + **draggable: true**: Allows the user to drag the modal.
     + **buttons: [...]**: Specifies an array of buttons for the modal. In this case, it includes a 'Close' button with an action to close the modal.
4. **else { alert('Failed to fetch order details. Please try again.'); }**: If the AJAX request is not successful, it shows an alert informing the user about the failure.
5. **error: function (xhr, status, error) { console.error('AJAX Error:', status, error); }**: This is a callback function that will be executed if there is an error in the AJAX request. It logs an error message to the console.

This function is designed to fetch data from the server, dynamically create an HTML form with default values, and display it in a Bootstrap modal using the BootstrapDialog library.

Top of Form

**Concerning PreparedStatements :**

A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency and protect against SQL injections.

The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.

The MySQL server supports using anonymous, positional placeholder with?.

Prepare is followed by execute. During execute the client binds parameter values and sends them to the server. The server executes the statement with the bound values using the previously created internal resources.

Here's a simple example:

1. <?php
2. mysqli\_report(MYSQLI\_REPORT\_ERROR | MYSQLI\_REPORT\_STRICT);  
   $mysqli= newmysqli("example.com", "user", "password", "database");  
     
   /\* Non-prepared statement \*/  
   $mysqli->query("DROP TABLE IF EXISTS test");  
   $mysqli->query("CREATE TABLE test(id INT, label TEXT)");  
     
   /\* Prepared statement, stage 1: prepare \*/  
   $stmt=$mysqli->prepare("INSERT INTO test(id, label) VALUES (?, ?)");  
     
   /\* Prepared statement, stage 2: bind and execute \*/  
   $id=1;  
   $label ='PHP';  
   $stmt->bind\_param("is", $id, $label); // "is" means that $id is bound as an integer and $label as a string  
     
   $stmt->execute();

We will present the PreparedStatement used in our code and explain this code in details:

 $update = "UPDATE products SET product\_name=?, description=?, img=?, updated\_at=NOW() WHERE id=?";

            $stmt = $conn->prepare($update);

            $stmt->bind\_param("sssi", $productName, $desc, $img, $productId);

            $stmt->execute();

            $stmt->close();

1.This SQL query is used to update the "products" table. It sets the "product\_name," "description," and "img" columns to the values specified by placeholders ("?"). Additionally, it updates the "updated\_at" column to the current timestamp. The condition for the update is based on the "id" column.

2.$stmt = $conn->prepare($update):This line uses the **prepare** method on the database connection (**$conn**) to create a prepared statement. Prepared statements are a security measure to prevent SQL injection attacks.

3.$stmt->bind\_param("sssi", $productName, $desc, $img, $productId):This line binds the placeholders in the SQL statement to actual values. The **"sssi"** parameter type string specifies that the values are strings (**s**) for "product\_name" and "description," and an integer (**i**) for the "id" column. The variables **$productName**, **$desc**, **$img**, and **$productId** hold the actual values that will be used in the query.

4.$stmt->execute():This line executes the prepared statement with the provided values.

5.$stmt->close():Finally, the **close** method is called to free up resources associated with the prepared statement.

**Concerning PDO :**

## 1. Introduction to PDO

PHP Data Objects (PDO) is an extension that provides a consistent interface to communicate with different databases by abstracting the differences between various database management systems (DBMS). PDO is not a database abstraction layer like other libraries such as Doctrine or Eloquent. Instead, it provides a standard API for working with databases, making it easier to switch between different DBMS without having to change your application code.

Some advantages of using PDO include:

* Consistent API for various database systems
* Support for prepared statements, which helps prevent SQL injection attacks
* Ability to use transactions and control error handling
* Improved performance over traditional PHP database extensions

## 2. PDO Connection

To connect to a database using PDO, you need to create a new PDO instance. The PDO constructor takes three arguments: the Data Source Name (DSN), the username, and the password. The DSN is a string containing the database driver, host, database name, and other optional parameters.

Here's an example of how to connect to a MySQL database using PDO:

try {

$dsn = "mysql:host=localhost;dbname=my\_database;charset=utf8mb4";

$username = "your\_username";

$password = "your\_password";

$pdo = new PDO($dsn, $username, $password);

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

echo "Connected successfully";

} catch (PDOException $e) {

echo "Connection failed: " . $e->getMessage();

In this example, we connect to a MySQL database named "my\_database" on the local host. The charset is set to utf8mb4 for proper handling of Unicode characters. The setAttribute method sets the error mode to throw exceptions, which allows for better error handling.

And here's an example of how to insert data into a table using a prepared statement:

try {

$pdo = new PDO($dsn, $username, $password);

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

$stmt = $pdo->prepare("INSERT INTO users (name, email) VALUES (:name, :email)");

$stmt->bindParam(':name', $name);

$stmt->bindParam(':email', $email);

$name = "John Doe";

$email = "john.doe@example.com";

$stmt->execute();

echo "New record created successfully";

} catch (PDOException $e) {

echo "Error: " . $e->getMessage();

}

In this example, we use placeholders :name and :email in the SQL query. The bindParam method binds the PHP variables $name and $email to the placeholders. The execute method runs the prepared statement with the bound values, safely inserting the data into the database.

## 3. Fetching Data

PDO provides several methods to fetch data from the database. The most common ones are fetch, fetchAll, and fetchColumn. These methods return the fetched data in various formats, such as arrays or objects.

$stmt=$pdo->prepare("SELECT \* FROM users");

$stmt->execute();

// Fetch a single row as an associative array

$row=$stmt->fetch(PDO::FETCH\_ASSOC);

// Fetch all rows as an array of associative arrays

$rows=$stmt->fetchAll(PDO::FETCH\_ASSOC);

// Fetch a single column's value from the first row

$name=$stmt->fetchColumn(0);

## 4. Transactions and Error Handling

Transactions are a way to group multiple database operations together, ensuring that either all operations succeed or none of them do. PDO provides methods like beginTransaction, commit, and rollback to work with transactions.

try{

$pdo->beginTransaction();

$stmt1=$pdo->prepare("INSERT INTO users (name, email) VALUES (:name, :email)");

$stmt1->execute([':name'=>'John',':email'=>'john@example.com']);

$stmt2=$pdo->prepare("INSERT INTO users (name, email) VALUES (:name, :email)");

$stmt2->execute([':name'=>'Jane',':email'=>'jane@example.com']);

$pdo->commit();

}catch(PDOException$e){

$pdo->rollback();

echo"Error: ".$e->getMessage();

}

In this example, we begin a transaction, execute two prepared statements, and then commit the transaction. If any error occurs during the execution of the statements, the transaction is rolled back, and no changes are made to the database.

## 5. Conclusion

In this course, we have covered the basics of the PHP PDO library, including setting up PDO, connecting to a database, using prepared statements, binding parameters, fetching data, and handling transactions and errors. Understanding and effectively using PDO is essential for any PHP developer working with databases.

We will present the PreparedStatement used in our code and explain this code in details:

 try {

        $pdo = newPDO("mysql:host=localhost;dbname=inventory", "root", "");//PDO:PHP Data Object

        //bdna l supplier l 5aso bl product l na2ayne

        $stmt = $pdo->prepare("SELECT supplier\_name FROM suppliers,

productsuppliers WHERE productsuppliers.product =$pidAND

productsuppliers.supplier = suppliers.id");

        $stmt->execute();

        $suppliers = $stmt->fetchAll(PDO::FETCH\_ASSOC);

        if ($suppliers) {

            $supplier\_arr = array\_column($suppliers, 'supplier\_name');

            $supplier\_list = implode(", ", $supplier\_arr);

        }

    } catch (PDOException$e) {

        die("Error: " . $e->getMessage());

    }

1. $pdo = new PDO("mysql:host=localhost;dbname=inventory", "root", ""):This line creates a new PDO instance to establish a connection to the MySQL database named "inventory" on the localhost. The username and password for the database are "root" (an empty password is used in this case).
2. $stmt = $pdo->prepare("SELECT supplier\_name FROM suppliers, productsuppliers WHERE productsuppliers.product = $pid AND productsuppliers.supplier = suppliers.id");$stmt->execute():Here, a SELECT statement is prepared using the **prepare** method. The SQL query retrieves the "supplier\_name" from the "suppliers" table based on the relationship with the "productsuppliers" table, using the product ID **$pid**. The **execute** method is then called to execute the prepared statement.
3. $suppliers = $stmt->fetchAll(PDO::FETCH\_ASSOC):The ‘**fetchAll’** method retrieves all rows from the result set as an associative array. Each element in the array corresponds to a row in the result set, with keys being column names.
4. **Processing Results:**If there are results (**$suppliers** is not empty), the code extracts the "supplier\_name" column values.
5. **Exception Handling:} catch (PDOException $e) {**

**die("Error: " . $e->getMessage());}:**This block catches any exceptions that may occur during the database operations and outputs an error message along with the error details using ‘**die’.**

**Thankyou for your time and hope everything is clear**