Working with APIs

- Creating a Simple JSON API with PHP
- Reading API Data with JavaScript
- Creating a Database-Driven API with PHP and MySQL
- Creating API Endpoints for Specific Data

Creating a Simple JSON API with PHP

In this tutorial, you will create a simple API using PHP.

The API will return JSON data when visited in a browser.

This is the same type of data format used by many real-world APIs.

Project Structure

Create a folder called:

```
api-demo
```

Inside the folder place:

```
api-demo
├── api.php
└── smart_security_data.json
```

Create the API File

Create:

```
api.php
```

Add:

```
<?php

header(
```

```
    "Content-Type: application/json"
);

echo file_get_contents(
    "smart_security_data.json"
);

?>
```

Save the file.

Understanding the Code

This line tells the browser that JSON data is being returned:

```
header(
    "Content-Type: application/json"
);
```

This line reads the JSON file:

```
file_get_contents(
    "smart_security_data.json"
);
```

This line sends the JSON data to the browser:

```
echo file_get_contents(
    "smart_security_data.json"
);
```

Test the API

Open:

```
http://localhost/api-demo/api.php
```

You should see:

```
{
  "homeId": "GC-HOME-014",
  "location": "Gold Coast"
}
```

along with the rest of the JSON data.

Test the API in a New Browser Tab

Notice that the URL now behaves like a data source.

Instead of displaying a webpage, it returns structured JSON data.

This is the same concept used by many modern APIs.

Example:

```
https://example.com/api/users
https://example.com/api/products
https://example.com/api/weather
```

Complete API File

```
<?php

header(
    "Content-Type: application/json"
);

echo file_get_contents(
    "smart_security_data.json"
);

?>
```

You have successfully created a JSON API using PHP.

Next tutorial: **Reading API Data with JavaScript**.

Reading API Data with JavaScript

In the previous tutorial, you created a simple API using PHP.

The API returns JSON data when visited in a browser.

In this tutorial, you will use JavaScript to read data from the API and display it on a webpage.

Project Structure

Your project should contain:

```
api-demo
├── api.php
├── index.html
└── smart_security_data.json
```

Create the HTML Page

Create:

```
index.html
```

Add:

```
<!DOCTYPE html>
<html>
<head>
    <title>API Demo</title>
</head>
```

```
<body>

<h1>API Dashboard</h1>

<div id="output"></div>

<script>

</script>

</body>
</html>
```

Save the file.

Read the API

Inside the `<script>` tags add:

```
fetch("api.php")
    .then(response => response.json())
    .then(data => {

        console.log(data);

    });
```

Save the file.

Test the API Request

Open:

```
http://localhost/api-demo/
```

Press:

```
F12
```

Open the **Console** tab.

You should see the JSON object displayed.

Display the Home Information

Replace:

```
console.log(data);
```

with:

```
document.getElementById(
    "output"
).innerHTML = `

    <p>
        <strong>Home ID:</strong>
        ${data.homeId}
    </p>

    <p>
        <strong>Location:</strong>
        ${data.location}
    </p>

`;
```

Refresh the page.

You should see:

```
Home ID: GC-HOME-014


Location: Gold Coast
```

Display Device Information

Replace the existing code with:

```
let html = `

    <h2>Devices</h2>

    <table border="1">

        <tr>
            <th>Device ID</th>
            <th>Device Type</th>
            <th>Room</th>
        </tr>

`;

data.devices.forEach(device => {

    html += `

        <tr>
            <td>${device.deviceId}</td>
            <td>${device.deviceType}</td>
            <td>${device.room}</td>
        </tr>

    `;

});

html += "</table>";

document.getElementById(
    "output"
).innerHTML = html;
```

Refresh the page.

You should now see a table of devices.

Why Use an API?

Previously, the data was loaded directly from:

```
fetch("smart_security_data.json")
```

Now the data is loaded from:

```
fetch("api.php")
```

This means the data can be:

- Generated dynamically
- Stored in a database
- Updated automatically
- Filtered before being sent

The webpage does not need to know where the data comes from. It only needs the JSON returned by the API.

Complete Page

```
<!DOCTYPE html>
<html>
<head>
    <title>API Demo</title>
</head>
<body>

<h1>API Dashboard</h1>

<div id="output"></div>

<script>

fetch("api.php")
```

```
    .then(response => response.json())
    .then(data => {

        let html = `

            <h2>Devices</h2>

            <table border="1">

                <tr>
                    <th>Device ID</th>
                    <th>Device Type</th>
                    <th>Room</th>
                </tr>

        `;

        data.devices.forEach(device => {

            html += `

                <tr>
                    <td>${device.deviceId}</td>
                    <td>${device.deviceType}</td>
                    <td>${device.room}</td>
                </tr>

            `;

        });

        html += "</table>";

        document.getElementById(
            "output"
        ).innerHTML = html;

    });

</script>
```

```
</body>
</html>
```

You have successfully consumed data from an API using JavaScript.

Next tutorial: **Creating a Database-Driven API with PHP and MySQL**.

Creating a Database-Driven API with PHP and MySQL

In the previous tutorials, the API returned data from a JSON file.

In this tutorial, you will create an API that reads data from a MySQL database and returns it as JSON.

The System Architecture

The completed system will work like this:

```
security_events table
        ↓
      api.php
        ↓
       JSON
        ↓
   JavaScript
```

Instead of reading a JSON file, the API will generate JSON directly from the database.

Create the API File

Create:

```
api_events.php
```

Add the database connection:

```
<?php
```

```
$conn = new mysqli(
    "localhost",
    "root",
    "",
    "project_db"
);

if ($conn->connect_error) {
    die("Connection failed: " .
        $conn->connect_error);
}

?>
```

Set the Content Type

Add:

```
header(
    "Content-Type: application/json"
);
```

The API will now return JSON instead of HTML.

Query the Database

Add:

```
$query = "
    SELECT *
    FROM security_events
";

$result =
    $conn->query($query);
```

This retrieves all records from the table.

Create an Array

Add:

```
$events = [];
```

This array will hold all database records.

Loop Through the Results

Add:

```
while (
    $row =
    $result->fetch_assoc()
) {

    $events[] = $row;

}
```

This converts each database record into an array item.

Convert the Array to JSON

Add:

```
echo json_encode(
    $events,
    JSON_PRETTY_PRINT
);
```

The completed API should now return JSON.

Test the API

Open:

```
http://localhost/api-demo/api_events.php
```

You should see:

```
[
    {
        "event_id":"1",
        "device_id":"CAM-01",
        "event_type":"motion_detected",
        "severity":"high"
    },
    {
        "event_id":"2",
        "device_id":"CAM-01",
        "event_type":"remote_access_attempt",
        "severity":"critical"
    }
]
```

The exact records will depend on your database.

Complete API File

```
<?php

$conn = new mysqli(
    "localhost",
    "root",
    "",
    "project_db"
```

```
);

if ($conn->connect_error) {
    die("Connection failed: " .
        $conn->connect_error);
}

header(
    "Content-Type: application/json"
);

$query = "
    SELECT *
    FROM security_events
";

$result =
    $conn->query($query);

$events = [];

while (
    $row =
    $result->fetch_assoc()
) {

    $events[] = $row;

}

echo json_encode(
    $events,
    JSON_PRETTY_PRINT
);

?>
```

Create a Webpage to Read the API

Create:

```
events.html
```

Add:

```
<!DOCTYPE html>
<html>
<head>
    <title>Security Events</title>
</head>
<body>

<h1>Security Events</h1>

<div id="output"></div>

<script>

fetch("api_events.php")
    .then(response => response.json())
    .then(data => {

        let html = `
            <table border="1">

                <tr>
                    <th>Device</th>
                    <th>Event Type</th>
                    <th>Severity</th>
                </tr>
        `;

        data.forEach(event => {

            html += `
                <tr>
```

```
                    <td>${event.device_id}</td>
                    <td>${event.event_type}</td>
                    <td>${event.severity}</td>
                </tr>
            `;

        });

        html += "</table>";

        document.getElementById(
            "output"
        ).innerHTML = html;

    });

</script>

</body>
</html>
```

Test the Complete System

Open:

```
http://localhost/api-demo/events.html
```

The page should display the records stored in MySQL.

Why This Is Useful

Previously:

```
JSON File
↓
JavaScript
```

Now:

```
MySQL Database
↓
PHP API
↓
JavaScript
```

This means:

- New records can be added to the database
- The API updates automatically
- The webpage updates automatically
- No JSON file needs to be edited manually

This is the foundation of many modern web applications.

Next tutorial: **Creating API Endpoints for Specific Data (Critical Events, Devices and Statistics)**.

Creating API Endpoints for Specific Data

In the previous tutorial, you created an API that returned every record from the `security_events` table.

In this tutorial, you will create multiple API endpoints that return different types of data.

This approach is commonly used in modern web applications.

Why Create Multiple Endpoints?

Your current API returns all records:

```
api_events.php
```

This works, but many applications need smaller, more focused datasets.

For example:

```
api_events.php
api_critical_events.php
api_statistics.php
```

Each endpoint has a specific purpose.

Create a Critical Events API

Create:

```
api_critical_events.php
```

Add:

```php
<?php

$conn = new mysqli(
    "localhost",
    "root",
    "",
    "project_db"
);

header(
    "Content-Type: application/json"
);

$query = "
    SELECT *
    FROM security_events
    WHERE severity = 'critical'
";

$result =
    $conn->query($query);

$events = [];

while (
    $row =
    $result->fetch_assoc()
) {

    $events[] = $row;

}

echo json_encode(
    $events,
    JSON_PRETTY_PRINT
);

?>
```

Test the Endpoint

Open:

```
http://localhost/api-demo/api_critical_events.php
```

You should only see critical events.

> **Screenshot Placeholder**
>
> Insert screenshot showing critical event results.

Create a Statistics API

Create:

```
api_statistics.php
```

Add:

```
<?php

$conn = new mysqli(
    "localhost",
    "root",
    "",
    "project_db"
);

header(
    "Content-Type: application/json"
);

$statistics = [
```

```
    "total_events" => 0,
    "critical_events" => 0,
    "high_events" => 0

];

$result =
    $conn->query(
        "SELECT COUNT(*) AS total
         FROM security_events"
    );

$statistics["total_events"] =
    $result->fetch_assoc()["total"];

$result =
    $conn->query(
        "SELECT COUNT(*) AS total
         FROM security_events
         WHERE severity='critical'"
    );

$statistics["critical_events"] =
    $result->fetch_assoc()["total"];

$result =
    $conn->query(
        "SELECT COUNT(*) AS total
         FROM security_events
         WHERE severity='high'"
    );

$statistics["high_events"] =
    $result->fetch_assoc()["total"];

echo json_encode(
    $statistics,
    JSON_PRETTY_PRINT
);
```

```
?>
```

Test the Statistics Endpoint

Open:

```
http://localhost/api-demo/api_statistics.php
```

Example:

```
{
    "total_events": 3,
    "critical_events": 1,
    "high_events": 1
}
```

Read the Statistics API with JavaScript

Create:

```
dashboard.html
```

Add:

```
<!DOCTYPE html>
<html>
<head>
    <title>Dashboard</title>
</head>
<body>

<h1>Security Dashboard</h1>
```

```
<div id="output"></div>

<script>

fetch("api_statistics.php")
    .then(response => response.json())
    .then(data => {

        document.getElementById(
            "output"
        ).innerHTML = `

            <p>
                Total Events:
                ${data.total_events}
            </p>

            <p>
                Critical Events:
                ${data.critical_events}
            </p>

            <p>
                High Events:
                ${data.high_events}
            </p>

        `;

    });

</script>

</body>
</html>
```

Common Endpoint Examples

Many real systems use endpoints such as:

```
/api/users
/api/products
/api/orders
/api/messages
/api/statistics
```

Each endpoint returns only the data required by the application.

Summary

You now have:

```
api_events.php
```

Returns all events.

```
api_critical_events.php
```

Returns only critical events.

```
api_statistics.php
```

Returns dashboard statistics.

This approach keeps APIs organised and makes applications easier to maintain.

Next tutorial: **Adding Parameters to an API Using URL Variables**.