Web Application Programming

Week 12. Advanced Front-End Concepts(Consuming APIs and asynchronous programming using Front-end build tools and package managers)

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com or jose@kumiuniversity.ac.ug

Agenda

1. Summary of previous lecture

2. Advanced Front-End Concepts(

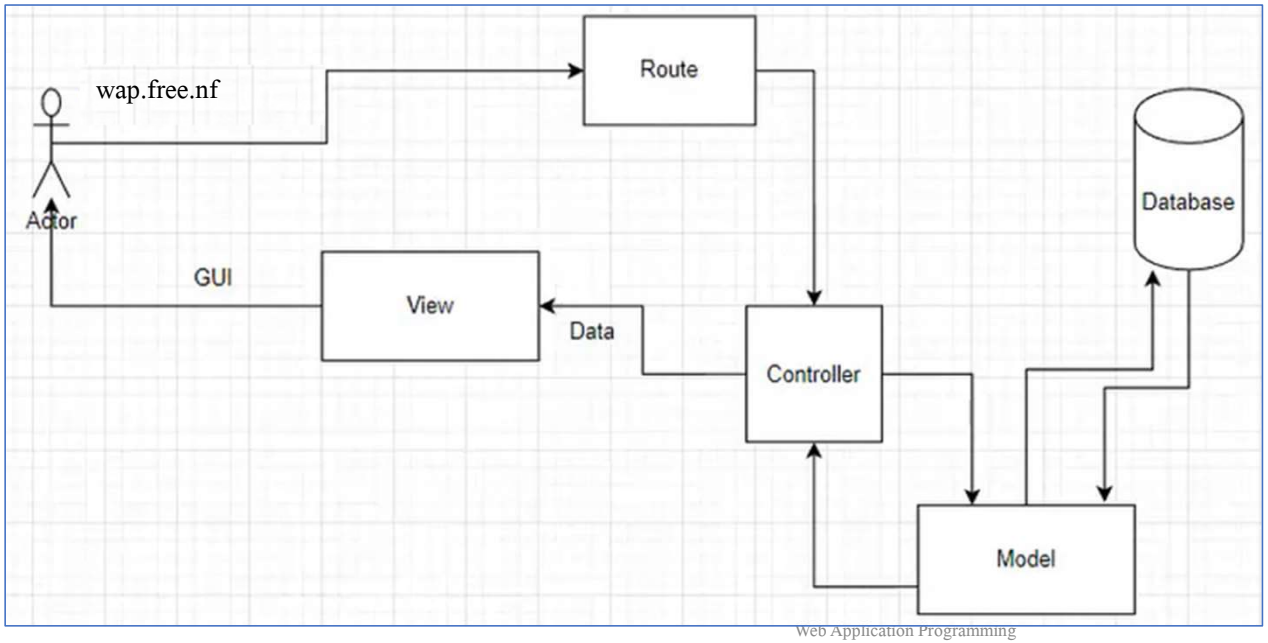
- Consuming APIs and asynchronous programming using Front-end build tools and package managers)

Summary of previous lecture

1. Advanced Front-End Concepts(
   i. Intro, recap on web application dev't frameworks,
   ii. Narrowed down to Laravel,
   iii. looked Laravel installation steps,
   iv. did the installation right from composer, Laravel, to creating Laravel Application project etc.)

Laravel Architecture

Laravel is a popular PHP web application framework known for its elegant syntax, expressive features, and robust architecture. Laravel follows the Model-View-Controller (MVC) architectural pattern, which separates the application into three main components:



Laravel Architecture

1. **Model:** The Model represents the data and business logic of the application. It is responsible for interacting with the database, performing data manipulation, and enforcing business rules. In Laravel, the Eloquent ORM (Object-Relational Mapping) is commonly used as the model layer, providing an easy and expressive way to interact with databases.

2. **Artisan Console:** Laravel comes with a command-line tool called Artisan. It provides various commands to help with common development tasks, such as database migrations, testing, and code generation.

Laravel Architecture

1. **View:** The View is responsible for presenting data to the user and handling user interfaces. In Laravel, Blade is the templating engine used for creating views. Blade provides a simple yet powerful syntax for writing templates and includes features like template inheritance and control structures.
2. **Controller:** The Controller acts as an intermediary between the Model and View. It processes user requests, interacts with the Model to retrieve or manipulate data, and passes the data to the View for presentation. Controllers in Laravel are typically responsible for handling HTTP requests, validating input, and coordinating the flow of data within the application.

Laravel Architecture

Beyond the MVC architecture, Laravel includes additional components and features that contribute to its overall architecture:

4. **Routing:** Laravel's routing system allows you to define how incoming HTTP requests should be handled. Routes are typically defined in the routes/web.php or routes/api.php files, mapping URLs to specific controller actions.

5. **Middleware:** Middleware in Laravel provides a mechanism to filter HTTP requests entering your application. Middleware can perform tasks such as authentication, logging, or modifying the request before it reaches the controller.

Laravel Architecture

6. Eloquent ORM: Eloquent is Laravel's ORM, providing an ActiveRecord implementation for interacting with databases. It allows you to define models and perform database operations using an expressive syntax.

7' Blade Templating Engine: Blade is a lightweight yet powerful templating engine in Laravel. It allows you to write templates with PHP code and provides features like template inheritance, sections, and components.

Laravel Architecture

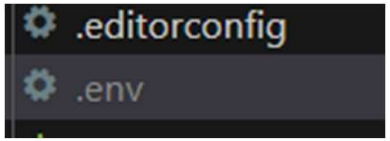
8. **Artisan Console:** Laravel comes with a command-line tool called Artisan. It provides various commands to help with common development tasks, such as database migrations, testing, and code generation.

Database crud operations using Laravel

Creating a table through Laravel

Creating database and changing Laravel default database user login details

1. Create database from phpMyAdmin of xampp
2. Go to **.env** file found inside your Laravel project, and locate database details
3. Locate database connection code
4. Change the database name: **myapp** to the one you created and like wise the username: **myapp** and password:**myapp**




```
11  DB_CONNECTION=mysql
12  DB_HOST=127.0.0.1
13  DB_PORT=3306
14  DB_DATABASE=myapp
15  DB_USERNAME=myapp
16  DB_PASSWORD=myapp
```

Create Table

5. Check if the connection to the database is okay: **php artisan migrate**

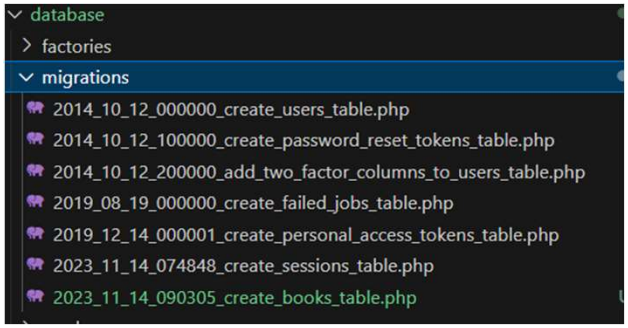
Creating a table.

Note. We do not create tables through phpMyAdmin, but use create directly using Laravel migration tool

Search for migration from Laravel website

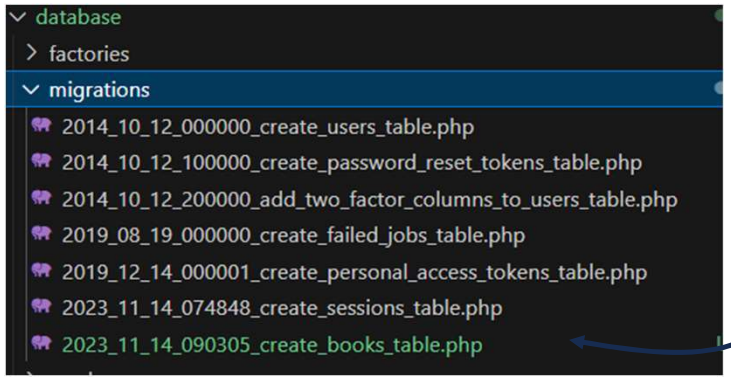
Scroll to Generating Migrations

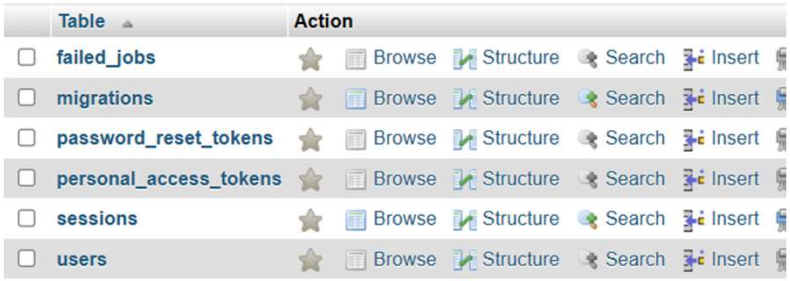
And click #Generate Migrations: php artisan make:migration create_books_table



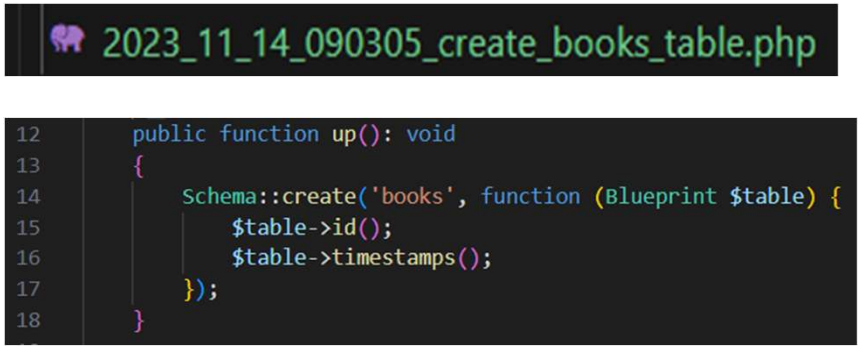
Access the file created

1. Check you C:\xampp\htdocs\myapp\databases\migrations to see if the database above is created.





1. You can open the books_table.php file created above, to add fields.



File before adding fields

Add fields to books table

```php
return new class extends Migration{
    // Run the migrations.

    public function up(): void{
        Schema::create('books', function (Blueprint $table) {
            $table->id();
            $table->string('bookname');
            $table->string('publisher');
            $table->string('author');
            $table->double('price');
            $table->date("yop");
            $table->timestamps();
        });
    }

    // Reverse the migrations.

    public function down(): void{
        Schema::dropIfExists('books');
    }
};
```

File after adding fields

```
PS C:\xampp\htdocs\myapp> php artisan migrate
```

myapp database after adding books table

	Table	Action
☐	books	☆ Browse
☐	failed_jobs	☆ Browse
☐	migrations	☆ Browse
☐	password_reset_tokens	☆ Browse
☐	personal_access_tokens	☆ Browse
☐	sessions	☆ Browse
☐	users	☆ Browse
	7 tables	Sum

```
SELECT * FROM `books`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

id	bookname	publisher	author	price	yop	created_at	updated_at

File after adding fields

Creating a model

To generate model, we use: `php artisan make:model books`

```
PS C:\xampp\htdocs\myapp> php artisan make:model
Books
```

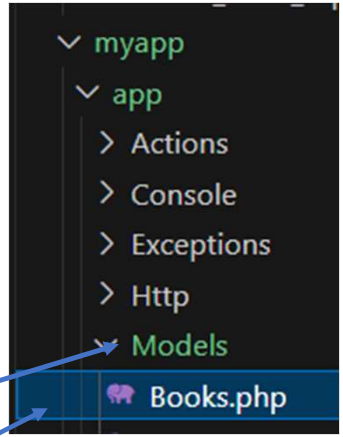
Feedback

```
   INFO  Model
[C:\xampp\htdocs\myapp\app/Models/Books.php]
created successfully.
```

Creating/Modifying the model

Accessing the model created

1. Click the **myapp** folder
2. Click on **app** sub folder
3. Click **Models**
4. Click **Books**.



Content before modification

myapp > app > Models > Books.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Books extends Model
{
    use HasFactory;
}
```

Entering data into the table through he model

myapp > app > Models > Books.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Books extends Model
{
    use HasFactory;
    protected $fillable =[
        'bookname',
        'publisher',
        'author',
        'price',
        'yob'
    ];
}

```

Model after entering data to be inserted into the books fields

Creating controller

User the following code to create a controller. `php artisan make:controller bookscontroler`

myapp > app > Http > Controllers > bookscontroler.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class bookscontroler extends Controller{
    //

}
```

Whenever you create a controller, it will come with predefined structure as seen in the figure above

Create a function inside the bookscontroler

Now let the index() inside bookscontroler return view(‘boos.index’);

myapp > app > Http > Controllers > bookscontroler.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class bookscontroler extends Controller{
    //
    public function index(){
        return view('books.index');
    }

}
```

Lets now create a view

1. Go to view folder
2. Created a new folder called Books
3. Create a php file called books.blade.php

Web Application Framework
Consuming APIs

An API, or Application Programming Interface, is a set of rules and protocols that allows one software application to interact with another. It defines the methods and data formats that applications can use to communicate with each other. APIs are used to enable the integration of different software systems, allowing them to work together and share data or functionality.

Here are key aspects of APIs:

1.Methods and Endpoints:

APIs provide a set of methods (functions or routines) that developers can use to perform specific actions. These methods are typically associated with specific endpoints (URLs or URIs) that represent different functionalities or resources.

Web Application Framework
Consuming APIs

Here are key aspects of APIs:

2. **Data Format:**

   APIs specify the data format for requests and responses. Common formats include JSON (JavaScript Object Notation) and XML (eXtensible Markup Language). JSON has become more prevalent due to its simplicity and readability.

3. **Authentication and Authorization:**

   APIs often include mechanisms for authentication and authorization to ensure that only authorized users or applications can access the provided functionality or data. This can involve API keys, OAuth tokens, or other authentication methods.

Web Application Framework
Consuming APIs

Here are key aspects of APIs:

3. **HTTP Methods:**

   Many APIs are built on the HTTP protocol and use standard HTTP methods like GET, POST, PUT, and DELETE to perform different actions. For example, a GET request might be used to retrieve data, while a POST request could be used to create new data.

4. **Documentation:**

   Good API design includes comprehensive documentation that explains how to use the API, including the available endpoints, request methods, required parameters, and expected responses. Documentation is crucial for developers to integrate with an API successfully.

Web Application Framework
Consuming APIs

Here are key aspects of APIs:

5. RESTful APIs:

REST (Representational State Transfer) is a widely used architectural style for designing networked applications. RESTful APIs adhere to REST principles, emphasizing stateless communication and the use of standard HTTP methods.

6.SOAP APIs:

SOAP (Simple Object Access Protocol) is another protocol for exchanging structured information in web services. Unlike REST, which often uses JSON and HTTP, SOAP typically uses XML and can be transported over various protocols.

Web Application Framework
Consuming APIs

Here are key aspects of APIs:

7. **Web APIs:**

   Web APIs, also known as web services or HTTP APIs, are APIs that are accessed over the web using standard web protocols. They play a crucial role in enabling communication between different web-based services.

APIs are fundamental to modern software development because they allow developers to leverage existing functionality, integrate with third-party services, and build on top of platforms and frameworks. They are used in a variety of contexts, including web development, mobile app development, cloud computing, and more.

Web Application Framework
Consuming APIs

Lets create a simple Application Programming interface for the following table called users found in wap database.

id	firstname	lastname	email	username	password
4	Elubu	Joseph	josebulinda@gmail.com	jose	202cb962ac59075b964b07152d234b70
10	Okello	Daniel	daniel@gmail.com	daniel	202cb962ac59075b964b07152d234b70
11	Max	ojang	ojang@gmail.com	ojang	202cb962ac59075b964b07152d234b70
12	Hong	Sekee	sekee@gmail.com	sekee	81dc9bdb52d04dc20036dbd8313ed055
13	Okello	Fred	fred@gmail.com	fred	202cb962ac59075b964b07152d234b70
14	Okele	okele	okele@gmail.com	okelel	202cb962ac59075b964b07152d234b70
15	T	T	T@gmail.com	T	202cb962ac59075b964b07152d234b70

Web Application Framework Consuming APIs-Sample code-1

connect.php | register.php | login.php | index.php | api.php | config.php | apiConsumer.php

```php
<?php

// Set headers to allow cross-origin resource sharing (CORS)
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

require 'connect.php';
// Check the HTTP method
$method = $_SERVER['REQUEST_METHOD'];

if ($method == 'GET') {
    // SQL query to fetch user details
    $sql = "SELECT id, firstname, lastname, email FROM users";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $users = array();

        // Fetch data and store it in an array
        while ($row = $result->fetch_assoc()) {
            $users[] = array(
                'id' => $row['id'],
                'firstname' => $row['firstname'],
                'lastname' => $row['lastname'],
                'email' => $row['email']
            );

        }
```

Web Application Framework
Consuming APIs-Sample code

```php
        // Return JSON response
        echo json_encode($users, JSON_PRETTY_PRINT);

    } else {
        // No results
        echo json_encode(array('message' => 'No users found.'));
    }
} else {
    // Invalid method
    echo json_encode(array('message' => 'Invalid request method.'));
}

// Close connection
$conn->close();

?>
```

Web Application Framework
Consuming APIs-Sample code

This PHP code is designed to create a simple API endpoint that retrieves user details from a MySQL database. Let's break down the code and understand how it works:

1. **CORS Headers**

```php
// Set headers to allow cross-origin resource sharing (CORS)
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
```

These lines set HTTP headers to allow Cross-Origin Resource Sharing (CORS), making it possible for the API to be accessed by web pages from different domains. The Content-Type header specifies that the response will be in JSON format.

Web Application Framework
Consuming APIs-Sample code

2. Database Connection:

```php
require 'connect.php';
```

This line includes the file connect.php, presumably containing the code to establish a connection to the database. It's common to separate database connection details into a separate file for reusability.

3. Check HTTP Method:

```php
$method = $_SERVER['REQUEST_METHOD'];
```

This line retrieves the HTTP request method (GET, POST, etc.) from the server's environment variables.

Web Application Framework
Consuming APIs-Sample code

4. Handle GET Request:

```php
if ($method == 'GET') {
    // ...
} else {
    // Invalid method
    echo json_encode(array('message' => 'Invalid request method.'));
}
```

The code checks if the HTTP method is GET. If it is, it proceeds to execute the code inside the if block; otherwise, it returns a JSON response indicating an invalid request method.

Web Application Framework
Consuming APIs-Sample code

5. Fetch User Details from Database:

```
$sql = "SELECT id, firstname, lastname, email FROM users";
$result = $conn->query($sql);
```

\These lines define a SQL query to select user details from a table named **users** with columns **id**, **firstname**, **lastname**, and **email**. The query is then executed, and the result is stored in the **$result** variable

Web Application Framework Consuming APIs-Sample code

6. Format Results and Return JSON:

```php
if ($result->num_rows > 0) {
    $users = array();
    while ($row = $result->fetch_assoc()) {
        $users[] = array(
            'id' => $row['id'],
            'firstname' => $row['firstname'],
            'lastname' => $row['lastname'],
            'email' => $row['email']
        );
    }
    echo json_encode($users, JSON_PRETTY_PRINT);
} else {
    // No results
    echo json_encode(array('message' => 'No users found.'));
}
```

If there are results from the database query, the code fetches each row, formats it as an associative array, and adds it to the $users array.

Finally, it uses json_encode to convert the array to a JSON string and echoes it as the API response. If there are no results, it returns a JSON response indicating that no users were found.

```php
$conn->close();
```

Web Application Framework
Consuming APIs-Sample code

Impotence of the above code

This code is useful for creating a simple API endpoint to expose user data in JSON format, allowing other applications or websites to consume this data.

The CORS headers enable the API to be accessed from different domains, making it suitable for use in web applications.

It follows a basic RESTful design, handling GET requests to retrieve data.

The separation of database connection code into a separate file (connect.php) promotes code organization and reusability.

Consuming APIs-Sample code-Output


```
[
    {
        "id": "4",
        "firstname": "Elubu",
        "lastname": "Joseph",
        "email": "josebulinda@gmail.com"
    },
    {
        "id": "10",
        "firstname": "Okello",
        "lastname": "Daniel",
        "email": "daniel@gmail.com"
    },
    {
        "id": "11",
        "firstname": "Max",
        "lastname": "ojang",
        "email": "ojang@gmail.com"
    },
    {
        "id": "12",
        "firstname": "Hong",
        "lastname": "Sekee",
        "email": "sekee@gmail.com"
    },
    {
        "id": "13",
        "firstname": "Okello ",
        "lastname": "Fred",
        "email": "fred@gmail.com"
    },
    {
        "id": "14",
        "firstname": "Okele",
        "lastname": "okele",
        "email": "okele@gmail.com"
    },
```

Consuming APIs-Sample code-Output


```
    {
        "id": "15",
        "firstname": "T",
        "lastname": "T",
        "email": "T@gmail.com"
    }
]
```

Consuming APIs-Sample consumer program

onnect.php | register.php | login.php | index.php | api.php | config.php | apiConsumer.php

```php
<?php

// Specify the API endpoint
$apiEndpoint = 'http://localhost/wap/api.php';

// Make a GET request to the API
$response = file_get_contents($apiEndpoint);

// Check if the request was successful
if ($response === FALSE) {
    die('Error accessing the API');
}

// Decode the JSON response
$data = json_decode($response, true);
```

This PHP code is a client-side script that interacts with a remote API. Let's break down each part of the code:

Consuming APIs-Sample consumer program

```php
// Check if decoding was successful
if ($data === NULL) {
    die('Error decoding JSON response');
}

// Check if the API returned an error message
if (isset($data['message'])) {
    echo 'API Error: ' . $data['message'];
} else {
    // API response was successful, process the data
    foreach ($data as $user) {
        echo 'ID: ' . $user['id'] . '<br>';
        echo 'Name: ' . $user['firstname'] . ' ' . $user['lastname'] . '<br>';
        echo 'Email: ' . $user['email'] . '<br><br>';
    }
}

?>
```

Consuming APIs-Sample consumer program explained

Specify the API Endpoint:

```
$apiEndpoint = 'http://localhost/wap/api.php';
```

This line sets the **$apiEndpoint** variable to the URL where the API is hosted. In this case, it's set to **http://localhost/wap/api.php**, assuming that's the correct URL for your API.

Make a GET Request to the API:

```
$response = file_get_contents($apiEndpoint);
```

The file_get_contents function is used to make a GET request to the specified API endpoint and store the response in the $response variable.

Consuming APIs-Sample consumer program explained

Check for Request Success:

```php
if ($response === FALSE) {
    die('Error accessing the API');
}
```

This checks if the file_get_contents operation was successful. If it returns FALSE, it means there was an error accessing the API, and the script terminates with an error message.

Decode JSON Response:

```php
$data = json_decode($response, true);
```

The json_decode function is used to convert the JSON response obtained from the API into a PHP associative array. The result is stored in the $data variable

Consuming APIs-Sample consumer program explained

Check for Decoding Success:

```php
if ($data === NULL) {
    die('Error decoding JSON response');
}
```

This checks if the JSON decoding operation was successful. If it returns NULL, there was an issue decoding the JSON, and the script terminates with an error message.

Check for API Error Message:

```php
if (isset($data['message'])) {
    echo 'API Error: ' . $data['message'];
} else {
    // API response was successful, process the data
}
```

This checks if the API response contains an error message. If an error message is present, it is echoed, and the script stops. Otherwise, it proceeds to process the data.

Consuming APIs-Sample consumer program explained

Process API Response:

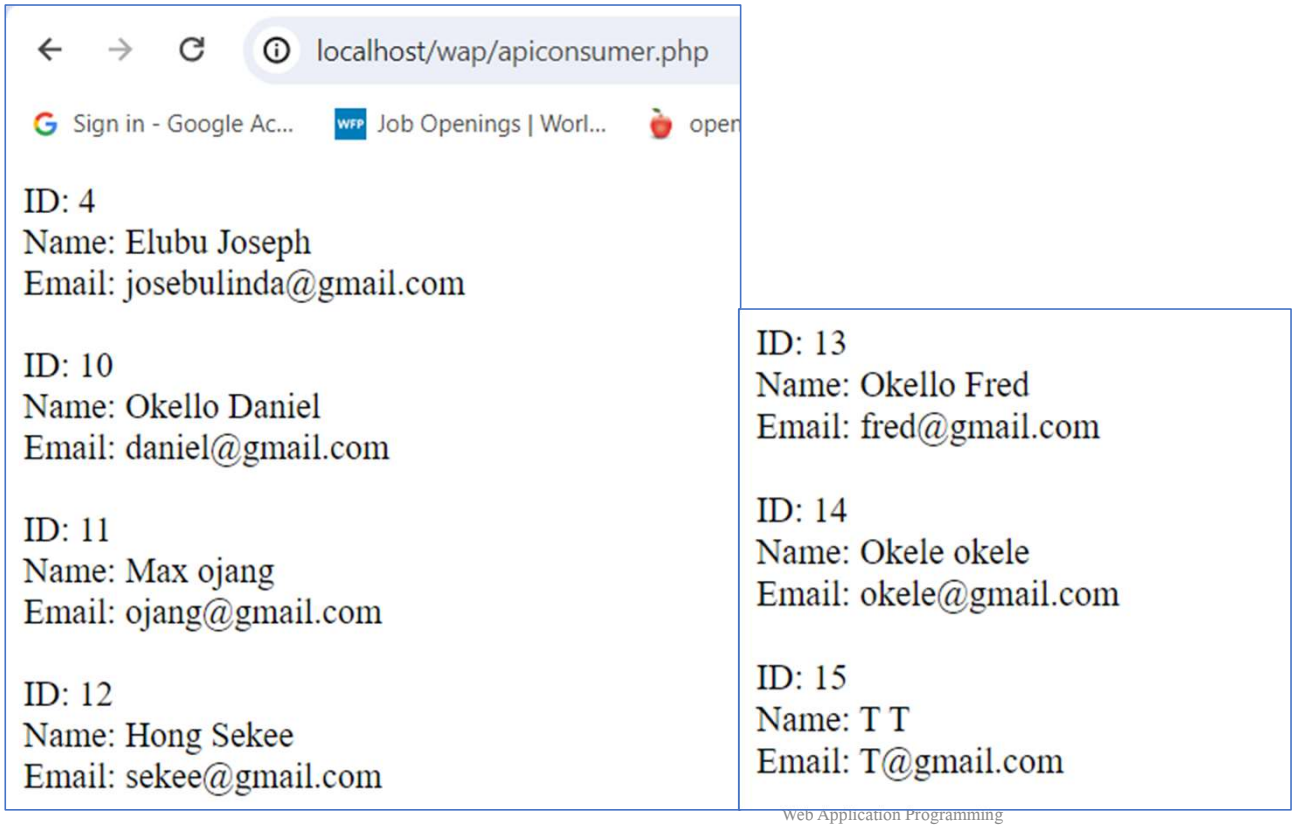
```php
foreach ($data as $user) {
    echo 'ID: ' . $user['id'] . '<br>';
    echo 'Name: ' . $user['firstname'] . ' ' . $user['lastname'] . '<br>';
    echo 'Email: ' . $user['email'] . '<br><br>';
}
```

If the API response was successful and didn't contain an error message, the script loops through the $data array (which should contain user details) and prints out information for each user, such as ID, Name, and Email.

Consuming APIs-Sample consumer program explained

In summary, the above PHP script fetches user details from a remote API, checks for potential errors in the process (e.g., connectivity issues, JSON decoding errors, or API error messages), and then processes and displays the retrieved data if the operation is successful.

Consuming APIs-Sample consumer program-output



Asynchronous Programming

Asynchronous programming

is a programming paradigm that allows tasks to be executed independently, without waiting for the completion of one task before starting another, Squashlabs. (2023).

In traditional synchronous programming, tasks are executed sequentially, and the program waits for each task to finish before moving on to the next one.

Asynchronous programming, on the other hand, enables tasks to be performed concurrently, improving efficiency and responsiveness.

Asynchronous programming

Key concepts and components related to asynchronous programming:

1.Callbacks:

Callbacks are functions passed as arguments to other functions. In the context of asynchronous programming, they are often used to specify what should happen when a task is completed. Callbacks are executed once the asynchronous operation finishes.

2.Promises:

Promises are a more structured way of handling asynchronous code. They represent the eventual completion or failure of an asynchronous operation and allow you to attach callbacks for success or failure. Promises help avoid the "callback hell" problem associated with deeply nested callbacks.

Asynchronous programming

Key concepts and components related to asynchronous programming:

3. Async/Await:

The async and await keywords were introduced in ECMAScript 2017 (ES8) to simplify asynchronous code. Functions marked with async return Promises, and the await keyword is used to wait for the resolution of a Promise. This makes asynchronous code look more like synchronous code, improving readability.

4. Event Loop:

The event loop is a fundamental concept in asynchronous programming. It's a mechanism that continuously checks the message queue for events and executes associated callback functions. This allows non-blocking execution of tasks.

Asynchronous programming

Key concepts and components related to asynchronous programming:

5. Non-blocking I/O:

Asynchronous programming is often associated with non-blocking I/O (Input/Output) operations. In a non-blocking system, a task that is waiting for I/O to complete doesn't block the entire program; instead, the program can continue executing other tasks in the meantime.

6. Concurrency vs. Parallelism:

Asynchronous programming enables concurrency, which is the ability to execute multiple tasks seemingly simultaneously. It's essential to note that concurrency is not the same as parallelism. Parallelism involves executing multiple tasks literally simultaneously, often requiring multiple processors or cores.

Asynchronous programming

Key concepts and components related to asynchronous programming:

7. Timers and Events:

Asynchronous programming frequently involves timers and events. Timers are used to schedule tasks to run after a specified amount of time, and events are used to respond to user actions or other external triggers.

Asynchronous programming sample code

Here's a simple example in JavaScript using Promises and async/await:

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data successfully fetched');
    }, 2000);
  });
}

async function fetchDataAsync() {
  try {
    const result = await fetchData();
    console.log(result);
  } catch (error) {
    console.error('Error:', error);
  }
}

fetchDataAsync();
```

In this example, **fetchData** returns a Promise that resolves after a simulated asynchronous operation (a 2-second delay).

The **fetchDataAsync** function uses **async/await** to handle the asynchronous code more elegantly.

Summary

Agenda

1. Summary of previous lecture

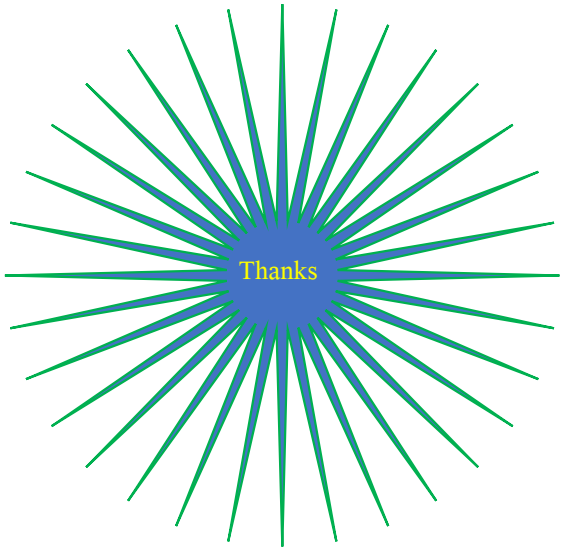
2. Advanced Front-End Concepts(

Loked Laravel structure-(Database, model, controller, view Route and other key aspects of the same)

- Created API-api.php and a program that Consumes from api.php API

Looked at asynchronous programming with sample code in JavaScript)

Thank you for Listening



References

The PHP framework for web artisans. Laravel. (n.d.). https://laravel.com/docs/10.x#getting-started-on-windows

Squashlabs. (2023, October 23). *How to implement asynchronous code in PHP*. Squash. https://www.squash.io/tutorial-implementing-asynchronous-code-in-php/#:~:text=In%20PHP%2C%20asynchronous%20programming%20can,program%20remains%20responsive%20and%20efficient.