**Introduction :**

In the rapidly evolving landscape of web technologies, integrating cloud services efficiently into web applications is pivotal for achieving scalability, security, and performance. This report outlines the implementation of a series of tasks designed to leverage Amazon Web Services (AWS) for building a robust web application. The tasks cover database management with DynamoDB, storage solutions with S3, and serverless computing using API Gateway and Lambda. The technologies selected for implementation, including Python, PHP, and Node.js, were chosen based on their compatibility with AWS services, ease of integration, and their widespread use in the industry. The following sections detail the execution of each task, highlighting the connection methods, programming languages used, and the rationale behind technology choices.

**Task 1: AWS DynamoDB**

* Created a "login" table in DynamoDB using AWS SDK for Python (Boto3). The script dynamically generated and inserted 10 entities. Python was chosen for its rich AWS support and ease of scripting. Connection to AWS was established using IAM credentials with appropriate permissions.
* Developed a Python script to create a "music" table and populate it from a JSON file using Boto3. The script handles both table creation and data insertion efficiently, leveraging Python's handling of data structures and AWS integration.

**Task 2: AWS S3**

* Utilized Python and Boto3 to automate the download of images from provided URLs and their upload to an S3 bucket. This choice was due to Python's effective handling of file operations and seamless integration with AWS services.

**Task 3: Login Page**

* The login functionality was implemented using PHP, interfacing directly with DynamoDB via AWS SDK for PHP. PHP was selected for its widespread use in web development and its native support for web page integration.

**Task 4: Register Page**

* The register page was also developed in PHP, using the AWS SDK to check and insert data into DynamoDB. PHP's server-side capabilities make it suitable for handling form data and interacting with databases.

**Task 5: Main Page**

* Implemented in PHP, the main page interacts with DynamoDB to fetch and manage music subscription data. PHP provides a straightforward approach for developing dynamic web content and handling session management.

**Task 6: API Gateway and Lambda Functions**

* Configured API Gateway to expose RESTful endpoints for user and subscription management. Lambda functions were written in Node.js due to its non-blocking I/O which is efficient for handling multiple I/O operations simultaneously, such as database reads/writes.
* Steps involved:
  1. Defined REST API resources and methods in API Gateway.
  2. Implemented Lambda functions in Node.js, which use AWS SDK to interact with DynamoDB. The functions handle registration, subscription, and unsubscription logic.
  3. Integrated Lambda functions with API Gateway, ensuring secure and scalable interactions between the web application and DynamoDB.
  4. Set up IAM roles for Lambda to securely access DynamoDB and other AWS services.
  5. Tested the setup to confirm that the web application's user interactions are processed correctly through the API endpoints.