**Overview**

In this project, we will create and deploy a serverless web application using AWS services: API Gateway, DynamoDB, and Lambda. This architecture allows us to build a scalable and cost-effective application without managing servers directly.

**Components**

1. **API Gateway**: Acts as the front door for our application, allowing clients to securely connect to our APIs created with Lambda functions.
2. **Lambda Functions**: These are the compute services that will execute our application logic in response to API requests. We will create several Lambda functions to handle different parts of our application.
3. **DynamoDB**: A fully managed NoSQL database service provided by AWS. We will use DynamoDB to store and retrieve data for our application.

**Steps to Implement**

**1. Designing the Application Architecture**

* **Identify Use Case**: Decide on a simple use case for your application. For example, a todo list manager or a note-taking app.
* **Define API Structure**: Design the API endpoints that your application will expose. For example:
  + /tasks (GET, POST): To list and create tasks.
  + /tasks/{id} (GET, PUT, DELETE): To get, update, and delete a specific task.

**2. Setting Up DynamoDB**

* **Create Tables**: Define the schema for DynamoDB tables based on your application needs. For instance, a Tasks table with taskId as the primary key.
* **Permissions**: Set up IAM roles and policies to allow Lambda functions to access DynamoDB.

**3. Implementing Lambda Functions**

* **Create Lambda Functions**: Write Lambda functions in your preferred programming language (Node.js, Python, etc.) to implement the API endpoints defined earlier.
* **Integration**: Configure Lambda functions to integrate with API Gateway so that they are invoked by API requests.

**4. Configuring API Gateway**

* **Create API**: Set up API Gateway with the endpoints defined in your application architecture.
* **Configure Integration**: Define how API Gateway should pass requests to your Lambda functions. This includes mapping request parameters and handling responses.

**5. Deploying and Testing**

* **Deploy APIs**: Deploy your API to a stage (e.g., prod, dev) in API Gateway.
* **Test Endpoints**: Use tools like Postman or curl to test your API endpoints and ensure they behave as expected.

**6. Monitoring and Scaling**

* **Monitoring**: Set up CloudWatch alarms to monitor API Gateway, Lambda functions, and DynamoDB for errors, latency, and other metrics.
* **Scaling**: AWS services like Lambda and DynamoDB automatically scale based on traffic, but ensure your design can handle increases in load.