# A Minor Project Synopsis on

Serverless Webpage

Submitted to Manipal University Jaipur

towards the partial fulfillment for the award of the degree of

# Bachelor of Technology in Computer & Communications Engineering

By

## Onkar Deokate

219303061

Section A

Under the Guidance of

**Ms. Anshika Malsaria**



# Department of Computer & Communication Engineering

# Manipal University Jaipur

## 2023-2024

**Synopsis**

### Introduction:

Introduction:

The "Serverless Web Application on AWS" project introduces an innovative exploration into the realm of serverless computing, aiming to showcase the power, scalability, and cost-effectiveness of Amazon Web Services (AWS) in web development. In this project, we embark on a journey to build a dynamic and responsive web application without the constraints of traditional server infrastructure. Serverless architecture represents a paradigm shift in web development by abstracting away server management concerns. This project aims to highlight the significance of this approach, emphasizing its impact on scalability, cost efficiency, and overall development agility. This introduction sets the stage for a detailed exploration of the project, encouraging developers and enthusiasts to delve into the exciting world of serverless web application development on AWS.

### Motivation:

This project is driven by the urgent need for scalable and secure solutions in modern web application development. Leveraging Amazon Web Services (AWS), we aim to showcase the efficiency of a multi-tier architecture, dividing the application into layers for enhanced flexibility and performance. The motivation extends to highlighting the capabilities of AWS services like EC2, Elastic Load Balancing, and RDS, emphasizing automation for deployment efficiency. Security considerations and cost optimization strategies further fuel the motivation, demonstrating the project's commitment to addressing the key challenges faced by contemporary web development.

### Project Objectives:

The primary objective of the "Serverless Web Application on AWS" project is to demonstrate the practical implementation and showcase the advantages of serverless architecture in developing a modern, scalable, and cost-effective web application. The project aims to achieve the following key objectives:

1. Illustrate Serverless Principles:

- Provide a comprehensive understanding of serverless principles, emphasizing the elimination of server management tasks, automatic scaling, and pay-as-you-go pricing models.

- Highlight the benefits of serverless architecture in terms of reduced operational overhead, increased development agility, and improved resource utilization.

2. Utilize AWS Services for Web Development:

- Showcase the integration of various AWS services to create a robust serverless infrastructure.

- Leverage AWS Lambda for serverless functions, Amazon S3 for static asset hosting, Amazon API Gateway for API management, AWS Cognito for user authentication, Amazon DynamoDB for data storage, Amazon CloudFront for global content delivery, AWS CloudWatch for monitoring, and AWS CodePipeline for continuous deployment.

3. Scalability and Performance:

- Emphasize the scalability advantages of serverless computing by designing the web application to automatically scale based on demand.

- Illustrate how serverless functions, combined with AWS services, enable the creation of a highly responsive and performant web application that can seamlessly handle varying levels of user traffic.

4. Cost Efficiency:

- Demonstrate the cost-effectiveness of serverless architecture by showcasing the pay-per-execution pricing model and resource optimization inherent in AWS Lambda.

- Highlight how serverless computing allows for precise cost control, as users only pay for the actual compute resources consumed during function execution.

5. Security and Authentication:

- Implement and showcase secure user authentication and authorization using AWS Cognito.

- Illustrate best practices for ensuring data security, both in transit and at rest, within the serverless web application.

6. Continuous Deployment and Development Workflow:

- Implement a streamlined continuous deployment workflow using AWS CodePipeline.

- Showcase how serverless architecture facilitates an efficient development lifecycle, allowing developers to deploy updates and new features seamlessly.

The overall objective of this project is to provide a practical and insightful example of building a serverless web application on AWS, empowering developers and businesses to harness the advantages of serverless computing for their own projects and applications.

### Methodology/ Planning of work:

The successful implementation of a serverless web application on AWS requires a well-defined methodology and planning. Below is a step-by-step guide outlining the methodology and planning of work for this project:

1. Define Project Scope and Objectives:

- Clearly outline the goals of the project, specifying features and functionalities.

- Define the target audience and expected user interactions with the web application.

2. Select Technologies and AWS Services:

- Choose the technologies and AWS services that align with project objectives.

- Identify the appropriate programming languages for Lambda functions and frontend development.

3. Design System Architecture:

- Create a detailed architecture diagram, illustrating the interaction between frontend, backend, and AWS services.

- Define the data flow, including user authentication, data storage, and communication between components.

4. Set Up AWS Environment:

- Create AWS accounts and set up necessary IAM roles with appropriate permissions.

- Configure AWS services such as Lambda, API Gateway, S3, DynamoDB, Cognito, CloudFront, CloudWatch, and CodePipeline.

5. Frontend Development:

- Develop the static frontend using HTML, CSS, and JavaScript.

- Integrate the frontend with AWS SDKs or API Gateway for seamless interaction with the backend.

6. Backend Development with AWS Lambda:

- Develop serverless functions using AWS Lambda to handle dynamic functionality.

- Define API endpoints using API Gateway to trigger Lambda functions.

7. Implement User Authentication and Authorization:

- Integrate AWS Cognito for secure user authentication and authorization.

- Implement proper identity management to control access to resources.

8. Data Storage with DynamoDB:

- Design and implement the data schema for DynamoDB.

- Develop Lambda functions to interact with DynamoDB for storing and retrieving data.

9. Global Content Delivery with CloudFront:

- Configure Amazon CloudFront to cache static assets at edge locations for faster content delivery.

- Optimize CloudFront settings for global scalability.

10. Monitoring and Logging with CloudWatch:

- Set up CloudWatch alarms and metrics for monitoring Lambda functions and other AWS resources.

- Implement logging to capture events and facilitate debugging.

11. Continuous Deployment with CodePipeline:

- Implement a CI/CD pipeline using AWS CodePipeline for automated testing and deployment.

- Ensure smooth and controlled release cycles.

12. Deployment and Monitoring:

- Deploy the web application to the production environment using the CI/CD pipeline.

- Monitor the application in the live environment, addressing any issues that may arise.

13. Optimization and Scaling:

- Optimize Lambda functions, DynamoDB queries, and other resources for cost efficiency.

- Implement auto-scaling mechanisms to handle varying workloads effectively.

By following this methodology, the development team can systematically plan, implement, and deploy a serverless web application on AWS, ensuring that it meets high standards of scalability, performance, and cost-effectiveness.

**TECHNOLOGY USED:**

The "Serverless Web Application on AWS" project utilizes a variety of technologies to build a dynamic and responsive web application. The chosen technologies encompass both frontend and backend components, leveraging the capabilities of Amazon Web Services (AWS) for a seamless serverless architecture. Below are the key technologies used in this project:

Frontend Technologies:

1. AWS SDK for JavaScript (AWS Amplify):

- The AWS SDK for JavaScript facilitates the interaction between the frontend and AWS services, enabling seamless integration with AWS resources such as AWS Cognito for authentication.

Backend Technologies:

3. AWS Lambda:

- Serverless compute service for executing backend functions without the need for managing servers. Used to handle dynamic functionality in response to API Gateway triggers.

4. Amazon API Gateway:

- A fully managed service that allows for the creation, publishing, and management of APIs. Used to define API endpoints that trigger AWS Lambda functions.

5. AWS Cognito

- A fully managed identity service that handles user authentication and authorization. Used to secure the application and manage user access.

6. Amazon DynamoDB:

- A fully managed NoSQL database service. Used for scalable and flexible data storage, allowing efficient retrieval and storage of application data.

Deployment and Continuous Integration:

7. AWS CodePipeline:

- A fully managed continuous delivery service that automates the build, test, and deployment phases of the release process. Used for setting up a continuous integration and deployment pipeline.

Content Delivery and Scalability:

8. Amazon CloudFront:

- A content delivery network (CDN) service that delivers static and dynamic web content with low latency and high transfer speeds. Used for global content delivery, caching static assets at edge locations.

Monitoring and Logging:

9. Amazon CloudWatch:

- A monitoring and management service that provides real-time insights into AWS resources. Used for monitoring Lambda functions, setting up alarms, and logging system events.

1. **Facilities required for proposed work:**

Key Points for Software:

1. Cloud Platform Access:

- Ensure access to an AWS account for leveraging cloud services in the development, deployment, and management of the multi-tier architecture.

2. Automation for Infrastructure:

- Utilize AWS CloudFormation or AWS CDK to implement infrastructure as code, automating the deployment and management of AWS resources.

3. Continuous Integration/Continuous Deployment (CI/CD):

- Implement CI/CD pipelines using tools like Jenkins or AWS CodePipeline to automate the build, testing, and deployment processes.

4. Security and Monitoring Measures:

- Implement robust security measures using AWS IAM, KMS for encryption, and monitoring tools such as AWS CloudWatch and CloudTrail.

Key Points for Hardware:

1. Development Machine Specifications:

- Ensure development machines meet the requirements for processing power, RAM, and storage capacity to support efficient coding and development tasks.

2. Networking Reliability:

- Maintain a reliable and high-speed internet connection for seamless access to AWS services and effective collaboration among team members.

3. Backup Power Solutions:

- Implement backup power solutions, such as Uninterruptible Power Supply (UPS), to prevent data loss or workflow disruptions during power outages.

### References

<https://aws.amazon.com/amplify/>

<https://aws.amazon.com/lambda/>

<https://aws.amazon.com/api-gateway/>

<https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html>

<https://aws.amazon.com/dynamodb/>

<https://aws.amazon.com/codepipeline/>

<https://aws.amazon.com/cloudfront/>

<https://aws.amazon.com/cloudwatch/>