MINOR PROJECT-II REPORT

On

Serverless Web Application on AWS

Submitted to department of computer science & engineering in partial fulfillment of the requirement for the Bachelor of Engineering 6th semester

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

Submitted By

Advaita Shukla (0208CS211015)
Archie Nagdev (0208CS211035)
Astha Kesharwani (0208CS211044)
Chhavi Sahu (0208CS211055)

विद्याऽमृतं मश्नुते

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Gyan Ganga College of Technology
Jabalpur, Madhya Pradesh
RAJIV GANDHI PRODYOGIKI VISHWAVIDYALAYA, BHOPAL (M.P.)
April 2024

PREFACE

Minor Project-II is an integral part of Bachelor of Technology and each and every student has to create the Minor Project-II in the 6th Semester while studying in the Institute.

This record is concerned about our practical Minor Project-II during 6th Semester i.e. third year of Bachelor of Technology course. We have taken our practical Minor Project-II in **Serverless Web Application on AWS** During this Minor Project-II, we got to learn many new things about the technology and its practical implementation. This Minor Project-II proved to be a milestone in our knowledge of the present environment. Everyday and every moment was an experience in itself, an experience which theoretical study can't provide.

ACKNOWLEDGEMENT

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior and acts during the course of study.

I express my sincere gratitude to our principal, Dr. Ajay Kumar Lala, Principal, Gyan Ganga College of Technology, Jabalpur for providing me an opportunity to undergo Minor Project-II in **Serverless Web Application on AWS.**

I am thankful to **Dr. *Vimmi Pandey***, HOD department of Computer Science and Engineering for her support, cooperation, and motivation provided to me during the Minor Project-II for constant inspiration, presence and blessings.

Thank you to my Mentor **Prof. Kamaljeet Singh Kalsi**, Project Guide, Department of Computer Science and Engineering. Thank you for seeing the potential and pushing to do our best. Your unwavering belief in our abilities has motivated us to achieve more than we ever thought possible.

I also extend my sincere appreciation to all faculties of, department of Computer Science and Engineering, GGCT who provided their valuable suggestions and precious time in accomplishing my Minor Project-II report.

Lastly, I would like to thank the almighty and my parents for their moral support and my friends with whom I shared my day to day experience and received lots of suggestions that my quality of work.

Astha Kesharwani (0208CS211044)
Archie Nagdev (0208CS211035)
Chhavi Sahu (0208CS211055)
Advaita Shukla (0208CS211015)

DECLARATION

We, Advaita Shulka (0208CS211015), Archie Nagdev (0208CS211035), Astha Kesharwani (0208CS211044),Chhavi Sahu (0208CS211055), B.Tech(Semester-VI) of the Gyan Ganga College of Technology, Jabalpur hereby declare that the Minor Project II Report entitled "**Serverless Web Application on AWS**" is an original work and data provided in the study is authentic to the best of my knowledge. This report has not been submitted to any other Institute for the award of any other degree.

Place: Jabalpur
Date :-

Advaita Shukla (0208CS211015)
Archie Nagdev (0208CS211035)
Astha Kesharwani (0208CS211015)
Chhavi Sahu (0208CS211055)

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Approved by:

Project Coordinator	**Project Supervisor**	**Head of Department**
Prof.Yogesh Tiwari	**Prof. kamaljeet singh**	**Dr. Vimmi Pandey**
Assistant Professor	**Assistant Professor**	**Department of CSE**
Department of CSE	**Department of CSE,**	**GGCT,JABALPUR**
GGCT,JABALPUR	**GGCT,JABALPUR**	

GYAN GANGA COLLEGE OF TECHNOLOGY
JABALPUR (MP)

Approved by AICTE New Delhi & Govt. of M.P.
(Affiliated to Rajiv Gandhi Prodyougiki Vishwavidhyalaya, Bhopal)

Certificate

This is to certify that the Minor Project-II report entitled "*Serverless Web Application*" is submitted by "*Chhavi Sahu (0208CS211055) , Advaita Shukla (0208CS211015), Astha Kesharwni (0208CS211044), Archie Nagdev (0208CS21135)* " for the partial fulfillment of the requirement for the 6th semester of Bachelor of technology in Department of Computer Science & Engineering from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P).

Project Coordinator	**Head of Department**
Prof.Yogesh Tiwari	**Dr. Vimmi Pandey**
Project Coordinator	**Department of CSE,**
Assistant Professor	**GGCT,JABALPUR**

Index

1. INTRODUCTION

1.1 PROBLEM IDENTIFICATION

Manual Management : The manual management of DynamoDB tables within the AWS environment consumes valuable time and is prone to human error. Each task, such as provisioning tables, configuring indexes, and updating settings, requires manual intervention, leading to inefficiencies and potential mistakes.

Lack of Automation: Current methods lack automation, which results in repetitive tasks and increases the risk of errors. Without automated processes to handle routine operations, developers and administrators spend significant time performing manual tasks that could be better utilized for more strategic activities.

Scalability Challenges: As applications grow in complexity and scale, managing DynamoDB tables manually becomes increasingly challenging. Scaling tables to handle growing workloads requires careful planning and execution, often leading to delays and performance issues.

Configuration Complexities: Configuring DynamoDB tables involves various parameters such as provisioned throughput, indexes, and table settings. Understanding and managing these configurations manually can be complex and error-prone, especially for users with limited experience or expertise.

1.2 PROPOSED SYSTEM

Our proposed system aims to address these challenges by developing a serverless web application using AWS Lambda, DynamoDB, and Amazon S3. Key features include:

Automation of common tasks : The application automated provisioning, configuration, and management tasks for DynamoDB tables.

Simplified configuration : Users can configure and manage DynamoDB tables through an intuitive web interface, reducing complexities.

Enhanced scalability : Leveraging serverless architecture, the application scales seamlessly based on demand, ensuring optimal performance as applications grow.

2. REQUIREMENT ANALYSIS AND MODELLING

2.1 OVERVIEW

Implementing automation and scalability solutions for DynamoDB in AWS is critical. By utilizing Infrastructure as Code (IaC), AWS SDKs/APIs, and serverless architectures, provisioning, configuration, and management tasks can be automated effectively.

Infrastructure as Code (IaC):

- Define DynamoDB resources in code:
    - Enables automated provisioning and configuration.
    - Facilitates version control for infrastructure changes.
    - Ensures consistency in resource deployment across environments.

AWS SDKs/APIs:

- Programmatically interact with DynamoDB:
    - Automate table creation, deletion, and updates.
    - Manage indexes, attributes, and throughput capacity.
    - Integrate DynamoDB operations into custom applications and workflows.

Serverless Architectures:

- Leverage AWS Lambda and serverless computing:
    - Execute code in response to DynamoDB events.
    - Scale seamlessly based on workload demand.
    - Reduce operational overhead by eliminating server management.

Monitoring and Cost Optimization:

- Utilize Amazon CloudWatch for monitoring:
    - Monitor DynamoDB performance metrics in real-time.
    - Set up alarms for specific performance thresholds.
- Optimize costs with:
    - On-demand capacity mode for variable workloads.
    - Auto scaling to adjust provisioned capacity based on demand.

Backup and Restore Automation:

- Implement automated backup and restore processes:
    - Use AWS Backup or custom scripts for backup scheduling.
    - Ensure data durability and quick recovery from incidents.
    - Automate backup retention policies and lifecycle management.

2.2 FUNCTIONAL REQUIREMENT

Scalability:
- Accommodate increasing data volumes and table counts without performance sacrifice.
- Ensures the system can handle growing workloads seamlessly.

Reliability:
- Maintains data integrity through accurate table operations.
- Guarantees reliable and consistent execution of automated processes.

Performance:
- Automation processes should not hinder swift request handling.
- System must maintain high performance levels during automation tasks.

Security:
- Stringent measures, including robust access controls, prevent unauthorized access or modifications.
- Data privacy and security must be maintained at all times.

Flexibility:
- Supports diverse table configurations and features.
- Enables customization and adaptation to specific business needs.

Monitoring and Logging:
- Tracks automation tasks and swiftly resolves issues.
- Allows real-time monitoring of system health and performance metrics.

Cost-effectiveness:
- Optimizes resource utilization to minimize expenses.
- Achieves desired performance levels and scalability efficiently.

Compliance:
- Ensures adherence to data privacy and security requirements.
- Meets regulatory obligations and maintains trust with stakeholders.

User-friendliness:
- Enhances ease of configuration and management.
- Increases usability and adoption among users.

Comprehensive Documentation:
- Aids understanding and effective utilization of the automation solution.
- Provides guidance on configuration, best practices, and troubleshooting.

2.3 SOFTWARE AND TOOLS REQUIREMENT

To streamline the management of DynamoDB tables within AWS, various software tools and methodologies can be leveraged, each offering unique advantages. AWS CloudFormation provides automation through declarative templates, simplifying the provisioning of DynamoDB tables alongside other resources. AWS CDK enables defining tables using familiar programming languages, promoting infrastructure as code practices. Terraform offers a versatile option for managing DynamoDB tables through its declarative configuration language, supporting AWS and facilitating resource management consistency.

AWS SDKs empower developers to interact with DynamoDB programmatically, facilitating automation of table operations. The Serverless Framework abstracts infrastructure complexities, facilitating the deployment and management of serverless applications, including DynamoDB tables. Integrating AWS Lambda allows for event-driven actions, such as table creation or updates. The AWS CLI streamlines DynamoDB management tasks directly from the command line, enhancing scripting and automation capabilities.

Additionally, third-party tools like Database and NoSQL Workbench offer specialized features and ease of use for DynamoDB management. Considerations like scalability, security, and cost optimization should guide tool selection to ensure efficient management of DynamoDB tables as applications evolve. By utilizing these tools and methodologies, organizations can effectively streamline DynamoDB table management within their AWS environments, fostering agility, reliability, and cost-effectiveness in application development and deployment.

2.4 HARDWARE REQUIREMENTS

Efficient management of NAND mode DB tables within the AWS environment demands meticulous consideration of hardware requirements. Compute resources are paramount, necessitating powerful CPUs like AWS EC2 instances with multiple cores to handle increasing workloads. Adequate memory (RAM) is vital for caching frequently accessed data, minimizing disk I/O, and enhancing overall performance, with requirements scaling based on database size and transaction volume.

Storage options, including Amazon EBS, S3, and Aurora, offer varying performance, durability, and cost considerations tailored to the database's needs. High-speed networking via Amazon VPC, Direct Connect, and Transit Gateway ensures seamless communication between instances, application servers, and clients, optimizing throughput and latency.

Robust backup and disaster recovery solutions, such as Amazon RDS snapshots and cross-region replication, safeguard data integrity and enable swift recovery in case of unforeseen events. Monitoring tools like CloudWatch and AWS Trusted Advisor, coupled with managed services like Amazon RDS and Aurora, streamline resource optimization and maintenance tasks, reducing administrative overhead.

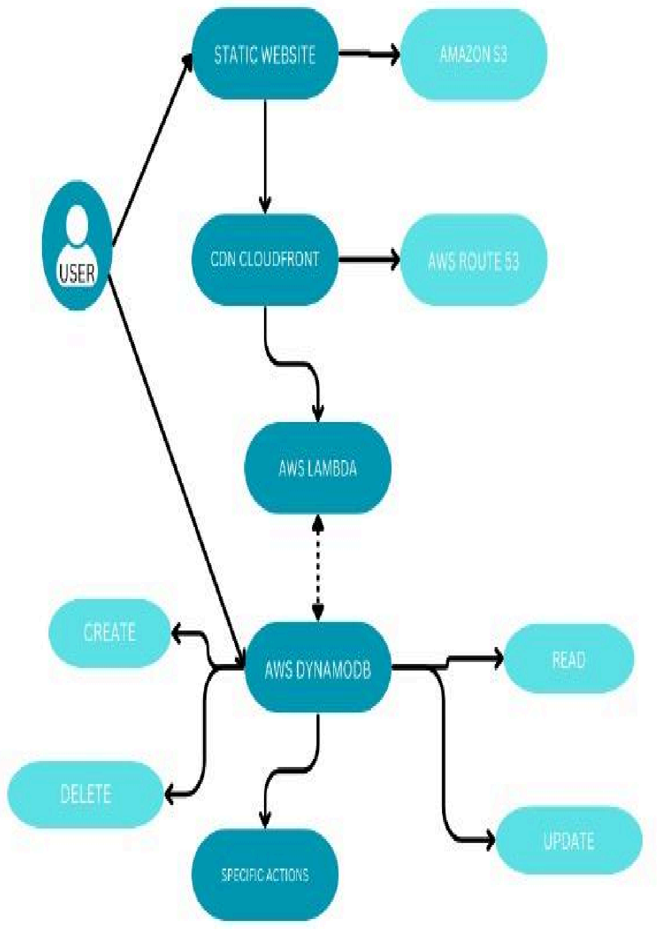
By meticulously addressing these considerations and harnessing AWS's scalable infrastructure and managed services, organizations can adeptly manage NAND mode DB tables, maximizing efficiency, scalability, and performance within the AWS ecosystem.

2.5 CONSTRAINTS

Managing DynamoDB tables manually within the AWS environment indeed presents several constraints. Firstly, manual provisioning demands significant time and effort, encompassing schema definition, index setup, and throughput capacity configuration. Secondly, scaling challenges arise, as adjusting capacity settings requires meticulous monitoring and manual adjustments, risking over-provisioning or degraded performance. Thirdly, human error in configuration poses risks like suboptimal performance or data loss. While AWS offers some automation features like auto-scaling, they may not cover all scenarios. As applications grow, managing multiple tables, ensuring consistency, and managing access controls becomes complex. Additionally, manual management may not fully leverage DynamoDB's scalability features, leading to inefficiencies. To alleviate these constraints, organizations adopt automation tools such as AWS CloudFormation or CDK. These tools enable infrastructure-as-code practices, automating provisioning, configuration, and management tasks. By embracing automation, organizations enhance consistency, scalability, and reliability while reducing the manual management burden. This approach ensures efficient DynamoDB utilization, optimal performance, and improved resource allocation in line with evolving workload demands.

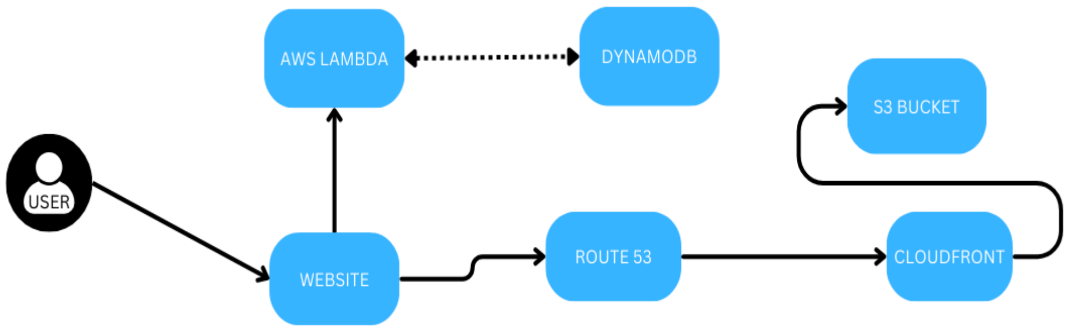
2.6 DIAGRAM

i) USE CASE DIAGRAM

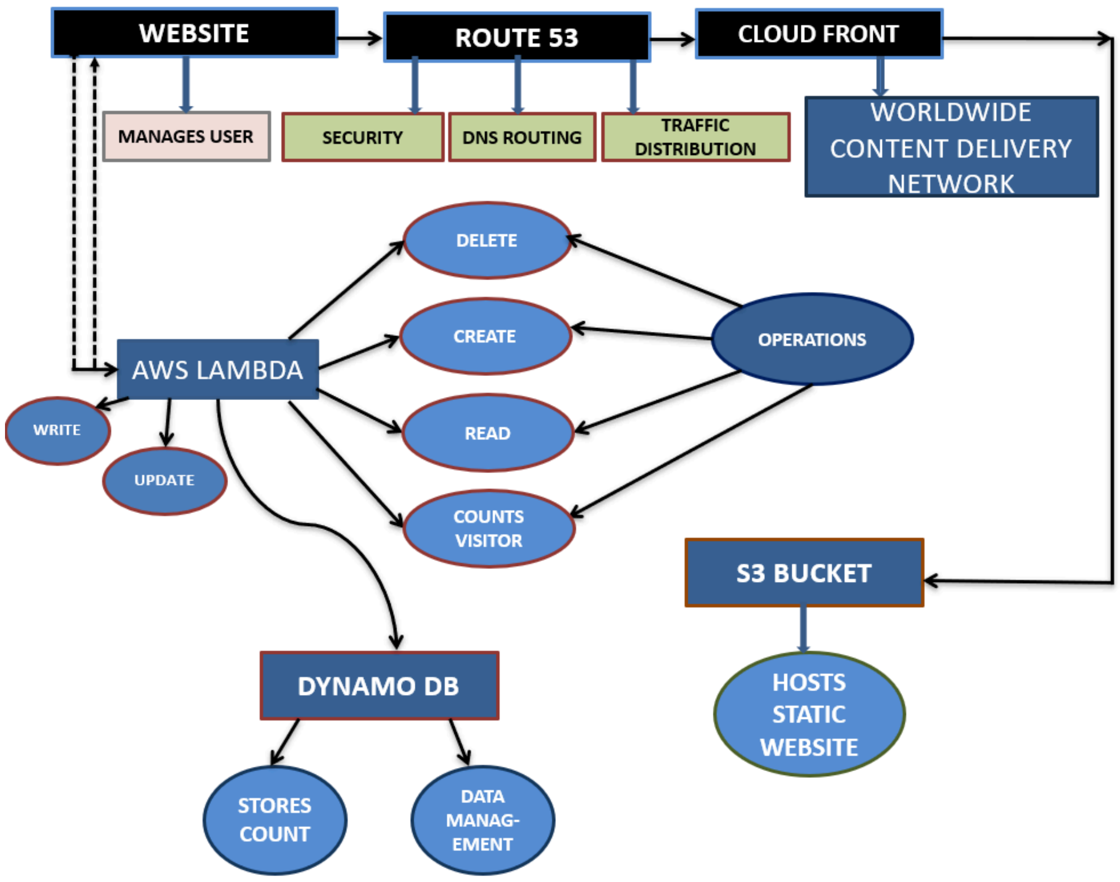


ii) DATA FLOW DIAGRAM

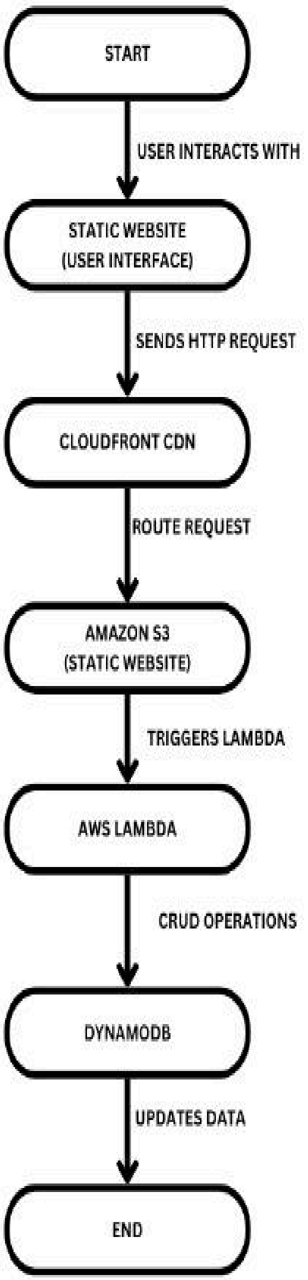
Level 0 Data Flow Diagram



Level 2 Data Flow Diagram



iii) ACTIVITY DIAGRAM



2.7 SOFTWARE DEVELOPMENT LIFE CYCLE MODEL USED

Implementing the Agile SDLC model ensures adaptability and collaboration throughout our project. Agile emphasizes iterative development, stakeholder involvement, and continuous improvement, fostering a dynamic and responsive development process. This approach enables us to deliver value to customers quickly while accommodating changes and feedback effectively.

Iterative Development: Agile divides the project into small iterations, allowing for frequent delivery of working software and incremental feature development.

Stakeholder Involvement: Regular feedback from stakeholders guides the development process, ensuring alignment with user needs and expectations.

Continuous Improvement: Agile promotes a culture of continuous improvement through regular retrospectives, where teams reflect on their processes and identify opportunities for enhancement.

Flexibility: Agile prioritizes flexibility and adaptability to changing requirements, allowing for adjustments and refinements throughout the development cycle.

Collaboration: Cross-functional teams collaborate closely, fostering communication and transparency to deliver high-quality software efficiently.

Customer-Centricity: Agile places a strong emphasis on delivering value to customers, with a focus on meeting their needs and delivering products that exceed expectations.

3. SOFTWARE DESIGN

3.1 OVERVIEW

The software design aims to automate the management of DynamoDB tables within the AWS environment to improve efficiency and scalability. It introduces a centralized system that leverages AWS SDKs and cloud services such as Lambda functions and Step Functions to automate common tasks like provisioning, scaling, monitoring, and updating DynamoDB tables based on usage patterns and defined policies. The design includes an easy-to-use web-based dashboard for users to configure automation settings and monitor table performance. This approach reduces human intervention, minimizes errors, and supports scaling operations smoothly as the application grows. Robust error handling and logging ensure reliability, while security best practices safeguard data integrity and access control .

3.2 SOFTWARE ARCHITECTURE

To address the challenges of managing DynamoDB tables manually within the AWS environment, a software architecture should focus on automation, scalability, and ease of use. The system should integrate with AWS SDKs and APIs to handle DynamoDB table creation, modification, and deletion automatically based on defined configurations and application requirements. A user-friendly interface, such as a web-based console or CLI, should enable users to manage and monitor tables efficiently. The architecture should support the specification of table parameters like partition keys, sort keys, and secondary indexes in configuration files. It should also include automated monitoring and alerting for performance, capacity, and error rates. Utilizing serverless architecture principles like AWS Lambda and Step Functions can enhance scalability and responsiveness. Moreover, best practices in security, such as least privilege access policies, should be applied to protect data and ensure compliance.

3.3 MODULES

A software design approach to manage DynamoDB tables efficiently within the AWS environment should involve automation, scalability, and performance optimization. The software design can be organized into key modules:

Configuration Module: Centralizes table settings such as read/write capacity modes, indexing, and data retention policies. Supports dynamic configuration management and enables changes to be applied programmatically and consistently across multiple tables.

Provisioning and Scaling Module: Automates table creation, deletion, and updates based on application needs. Utilizes AWS SDKs and APIs to handle table operations efficiently, including scaling read and write capacities dynamically in response to workload variations.

Data Management Module: Manages data operations like CRUD (Create, Read, Update, Delete) transactions and batch writes/reads, providing a layer of abstraction over DynamoDB's API for easier data manipulation. Includes error handling and retry logic to ensure robust operations.

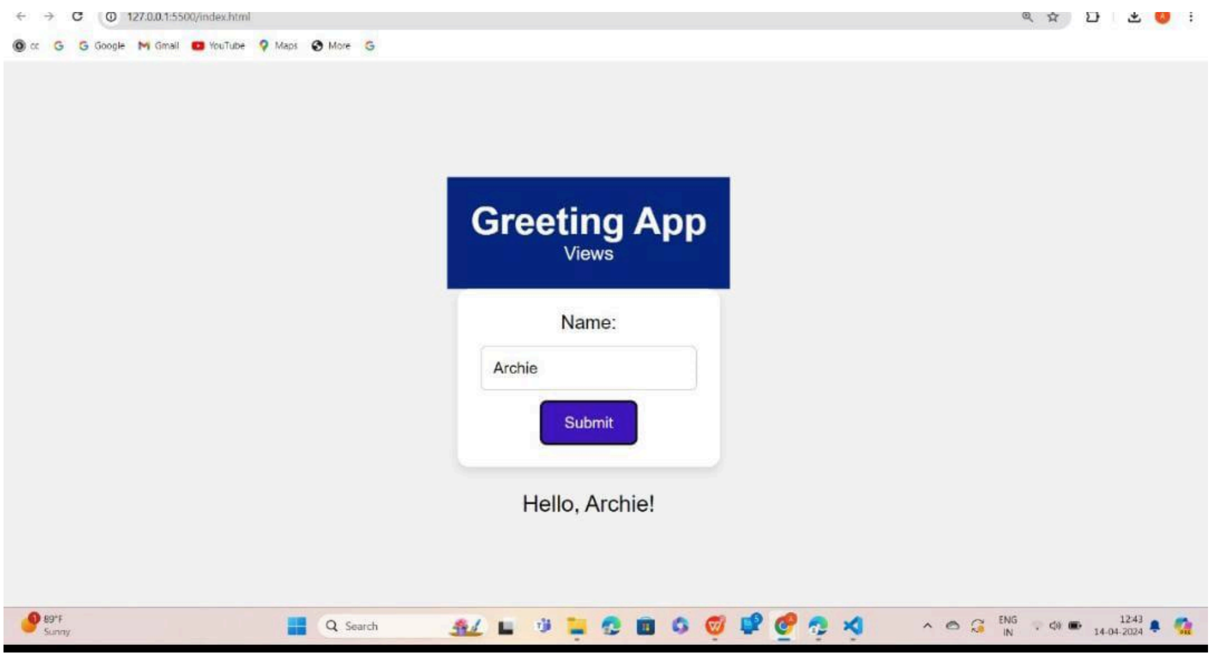
Monitoring and Analytics Module: Integrates with AWS CloudWatch and other monitoring tools to track performance metrics and usage patterns. Provides insights for capacity planning, bottleneck identification, and optimization opportunities.

Backup and Restore Module: Automates data backup processes using DynamoDB's point-in-time recovery and on-demand backup features. Manages backup schedules and restoration procedures to maintain data integrity and availability.

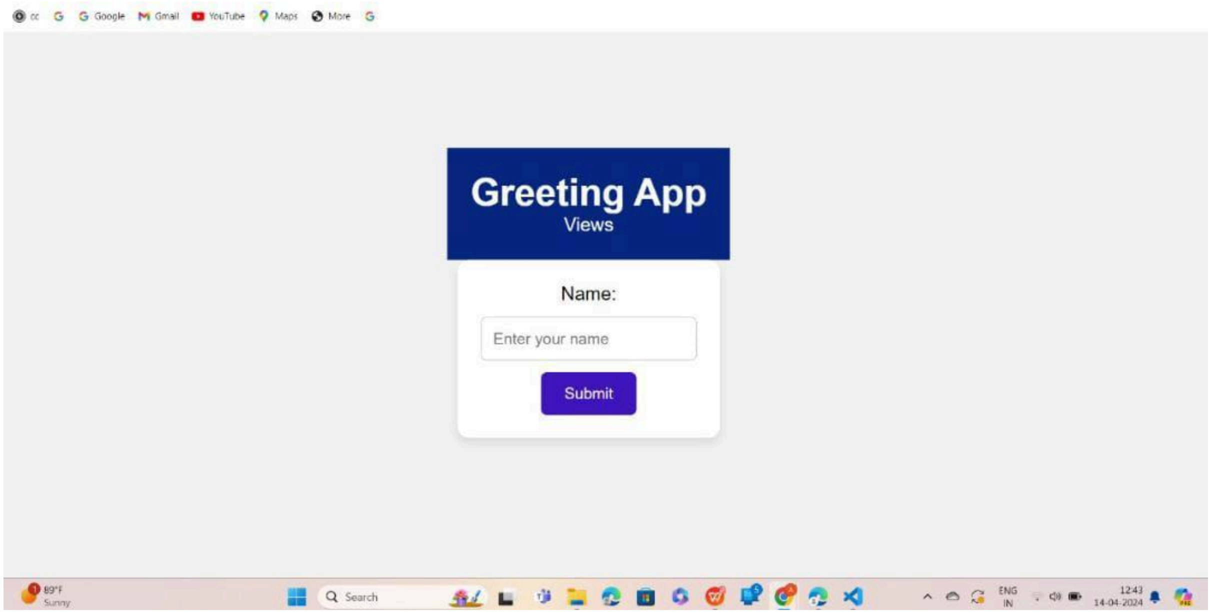
Security and Access Control Module: Implements security best practices such as data encryption, fine-grained access control using IAM policies, and auditing capabilities to ensure data protection and compliance.

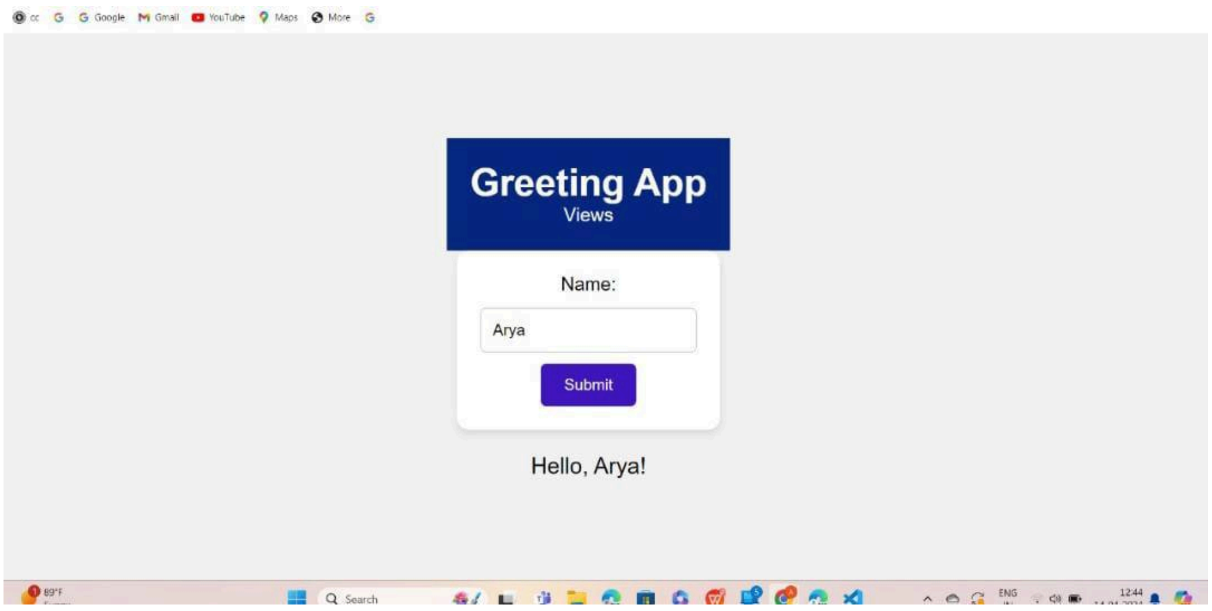
By designing the software with these modules, DynamoDB management becomes streamlined, scalable, and error-resistant, enabling more efficient handling of table operations and improved overall application performance.

3.4 GUI



Hello, Archie!

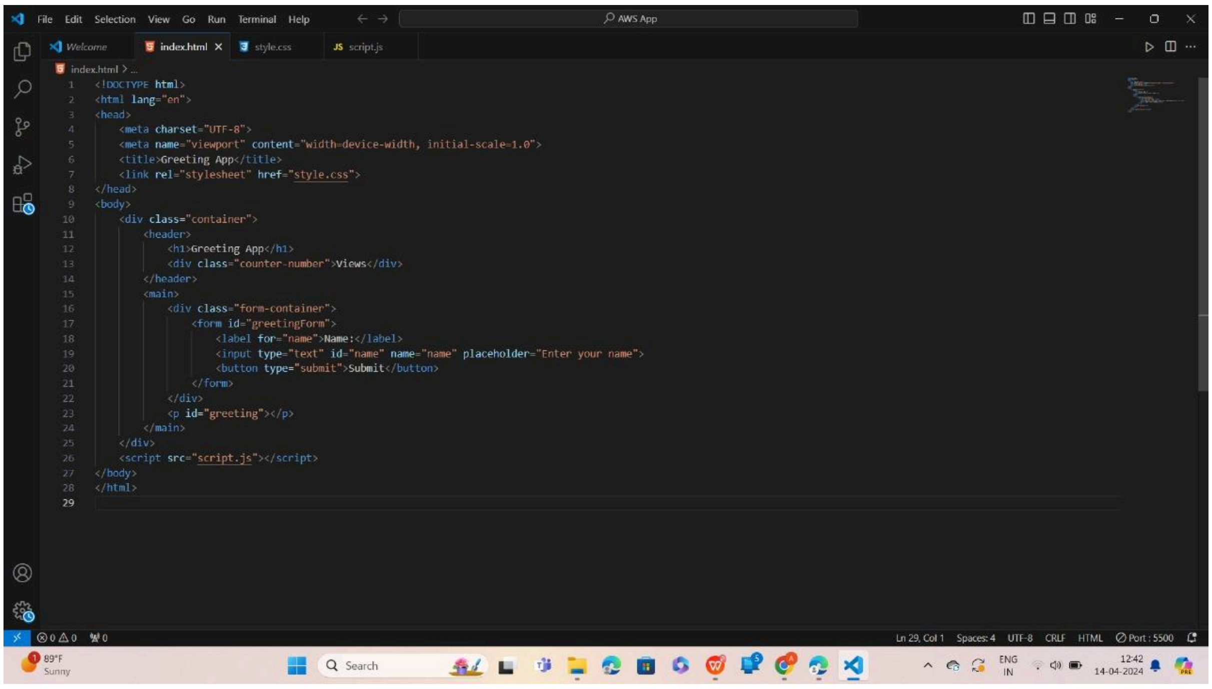




3.5 DATA DESIGN

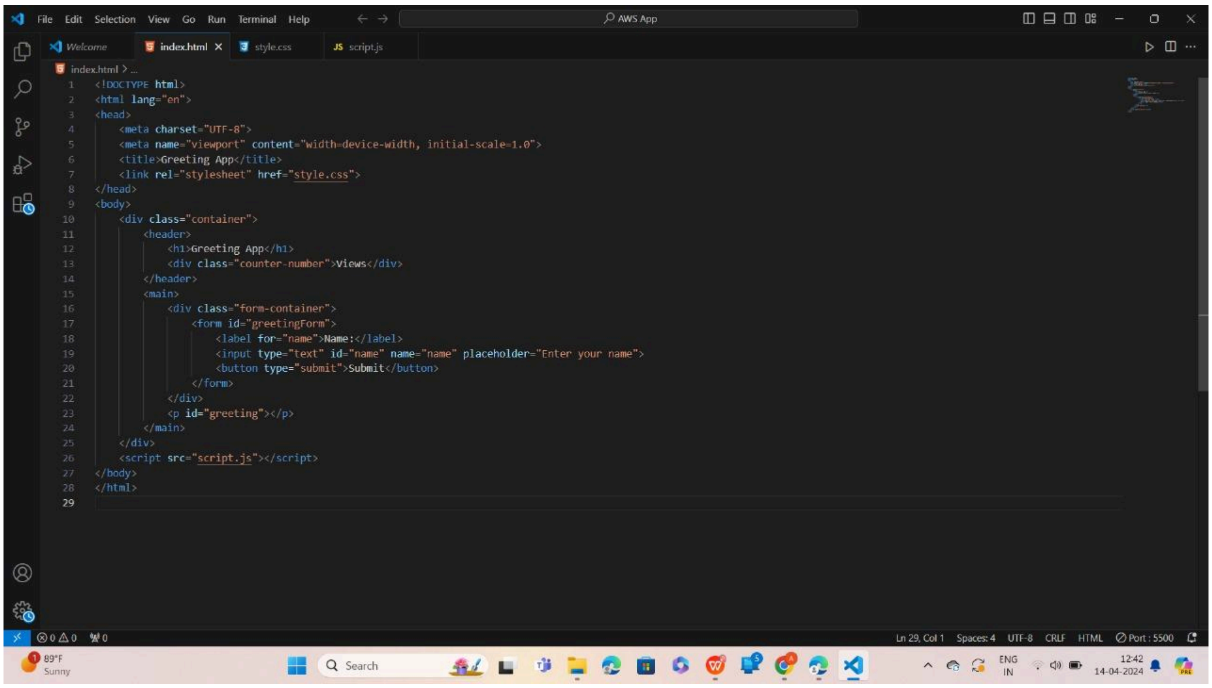
When managing DynamoDB tables manually within the AWS environment, challenges such as increased time consumption and a higher likelihood of human error can arise, impacting efficiency and scalability as applications grow. These methods often involve manually provisioning tables, setting configurations, and managing data throughput, making them labor-intensive and potentially error-prone tasks. To address these limitations, a software design approach should focus on automation and streamlined data design. This involves utilizing Infrastructure as Code (IaC) frameworks like AWS CloudFormation or Terraform to automate the provisioning and management of DynamoDB tables, ensuring consistent configurations and facilitating version control. Additionally, implementing dynamic scaling policies based on real-time monitoring can optimize table performance and cost-effectiveness. A well-structured data design, including the thoughtful selection of partition and sort keys, can improve query efficiency and data retrieval speeds. By integrating these strategies, the overall efficiency and scalability of DynamoDB management can be significantly enhanced, allowing applications to grow without hindering performance.

3.6 SOURCE CODE

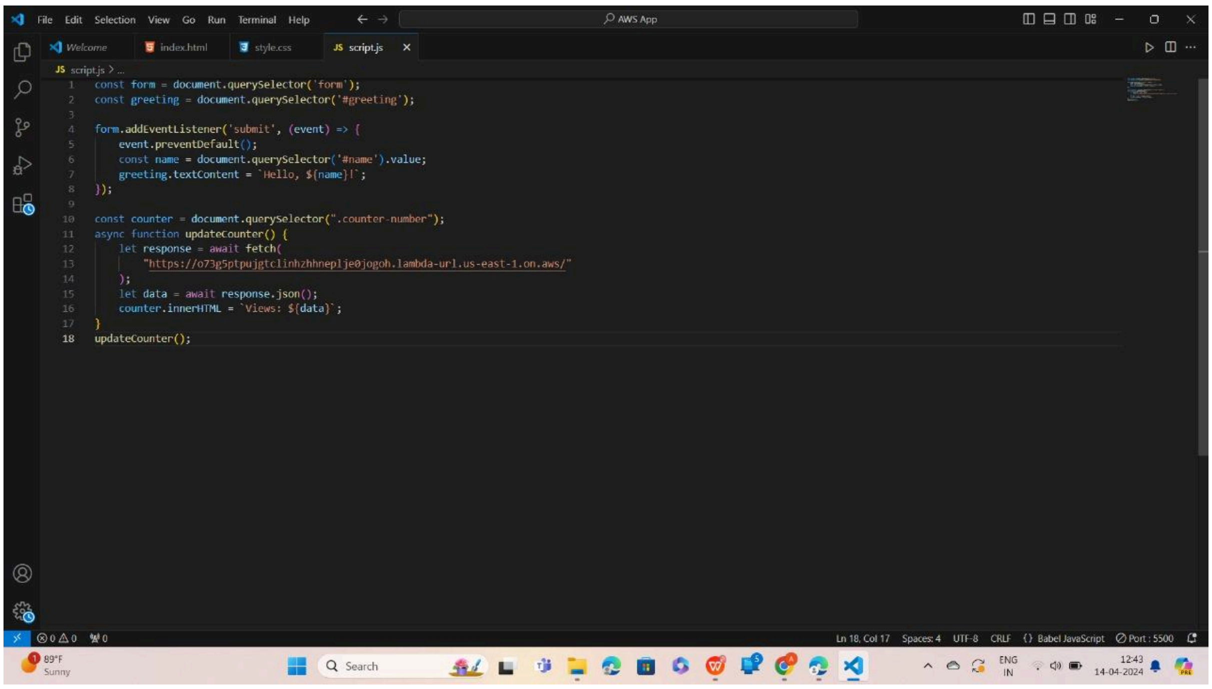


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Greeting App</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Greeting App</h1>
            <div class="counter-number">Views</div>
        </header>
        <main>
            <div class="form-container">
                <form id="greetingForm">
                    <label for="name">Name:</label>
                    <input type="text" id="name" name="name" placeholder="Enter your name">
                    <button type="submit">Submit</button>
                </form>
            </div>
            <p id="greeting"></p>
        </main>
    </div>
    <script src="script.js"></script>
</body>
</html>

```



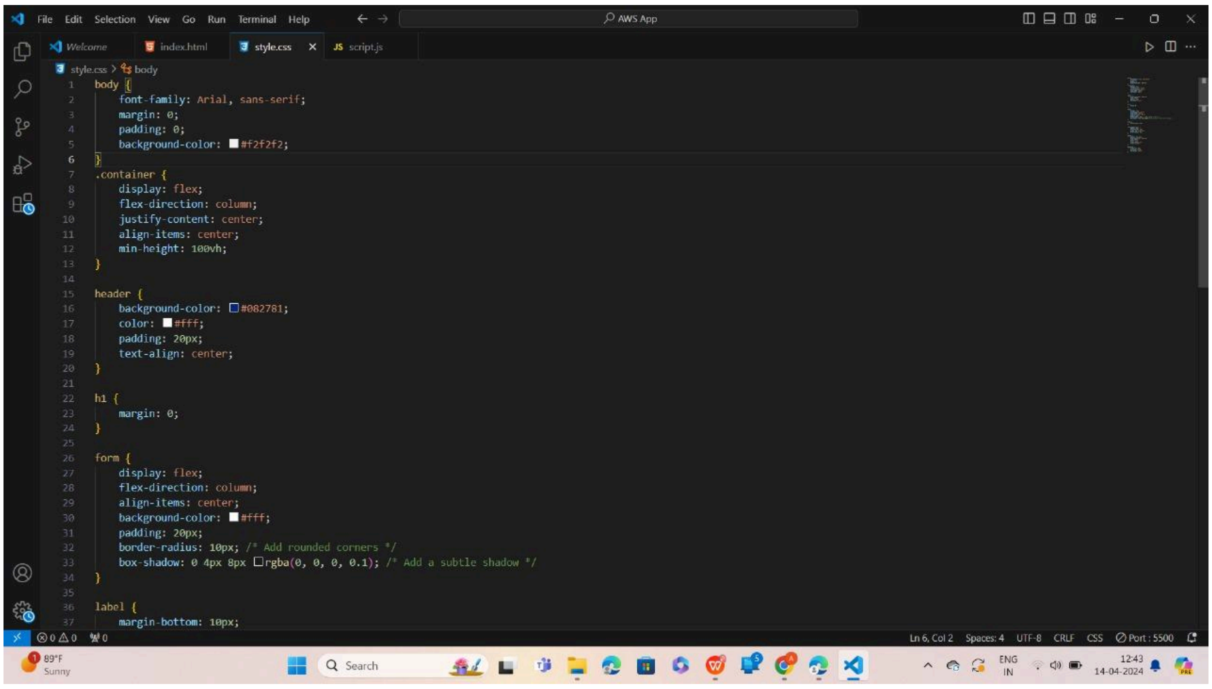
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Greeting App</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Greeting App</h1>
            <div class="counter-number">Views</div>
        </header>
        <main>
            <div class="form-container">
                <form id="greetingForm">
                    <label for="name">Name:</label>
                    <input type="text" id="name" name="name" placeholder="Enter your name">
                    <button type="submit">Submit</button>
                </form>
            </div>
            <p id="greeting"></p>
        </main>
    </div>
    <script src="script.js"></script>
</body>
</html>
```



```javascript
const form = document.querySelector('form');
const greeting = document.querySelector('#greeting');

form.addEventListener('submit', (event) => {
    event.preventDefault();
    const name = document.querySelector('#name').value;
    greeting.textContent = `Hello, ${name}!`;
});

const counter = document.querySelector(".counter-number");
async function updateCounter() {
    let response = await fetch(
        "https://o73g5ptpujgtclinhzhhneplje0jogoh.lambda-url.us-east-1.on.aws/"
    );
    let data = await response.json();
    counter.innerHTML = `Views: ${data}`;
}
updateCounter();
```


File Edit Selection View Go Run Terminal Help
AWS App
Welcome
index.html
style.css
script.js
style.css > body
Ln 6, Col 2 Spaces: 4 UTF-8 CRLF CSS Port : 5500
89°F Sunny
Search
ENG IN
12:43 14-04-2024

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f2f2f2;
}
.container {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
}

header {
    background-color: #082781;
    color: #fff;
    padding: 20px;
    text-align: center;
}

h1 {
    margin: 0;
}

form {
    display: flex;
    flex-direction: column;
    align-items: center;
    background-color: #fff;
    padding: 20px;
    border-radius: 10px; /* Add rounded corners */
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Add a subtle shadow */
}

label {
    margin-bottom: 10px;
```

3.5 DATA DESIGN

In designing the data structure for our serverless web application, we prioritize efficiency, scalability, and ease of use. Our approach focuses on creating a well-organized schema that accommodates the application's requirements while optimizing performance and resource utilization. Here's how we've designed the data:

Table Design:

- We define clear and concise table structures for storing different types of data, ensuring separation of concerns and data integrity.
- Each table is designed to serve a specific purpose, minimizing redundancy and improving query performance.

Primary Keys and Indexes:

- We carefully choose primary keys and secondary indexes to support efficient data retrieval and querying.
- Primary keys are selected based on access patterns and query requirements, ensuring optimal data access.

Data Modeling:

- We employ normalization and denormalization techniques based on the application's read and write patterns.
- Normalization reduces data redundancy and ensures consistency, while denormalization enhances query performance and reduces join operations.

Partitioning Strategy:

- We implement a partitioning strategy to distribute data evenly across DynamoDB partitions, preventing hot partitions and ensuring scalability.

- Partition keys are chosen strategically to distribute data evenly and minimize contention.

Data Validation and Constraints:

- We enforce data validation and constraints at the application level to maintain data integrity and prevent invalid data entry.
- Constraints are defined to ensure that only valid and consistent data is stored in the database.

Optimized Queries:

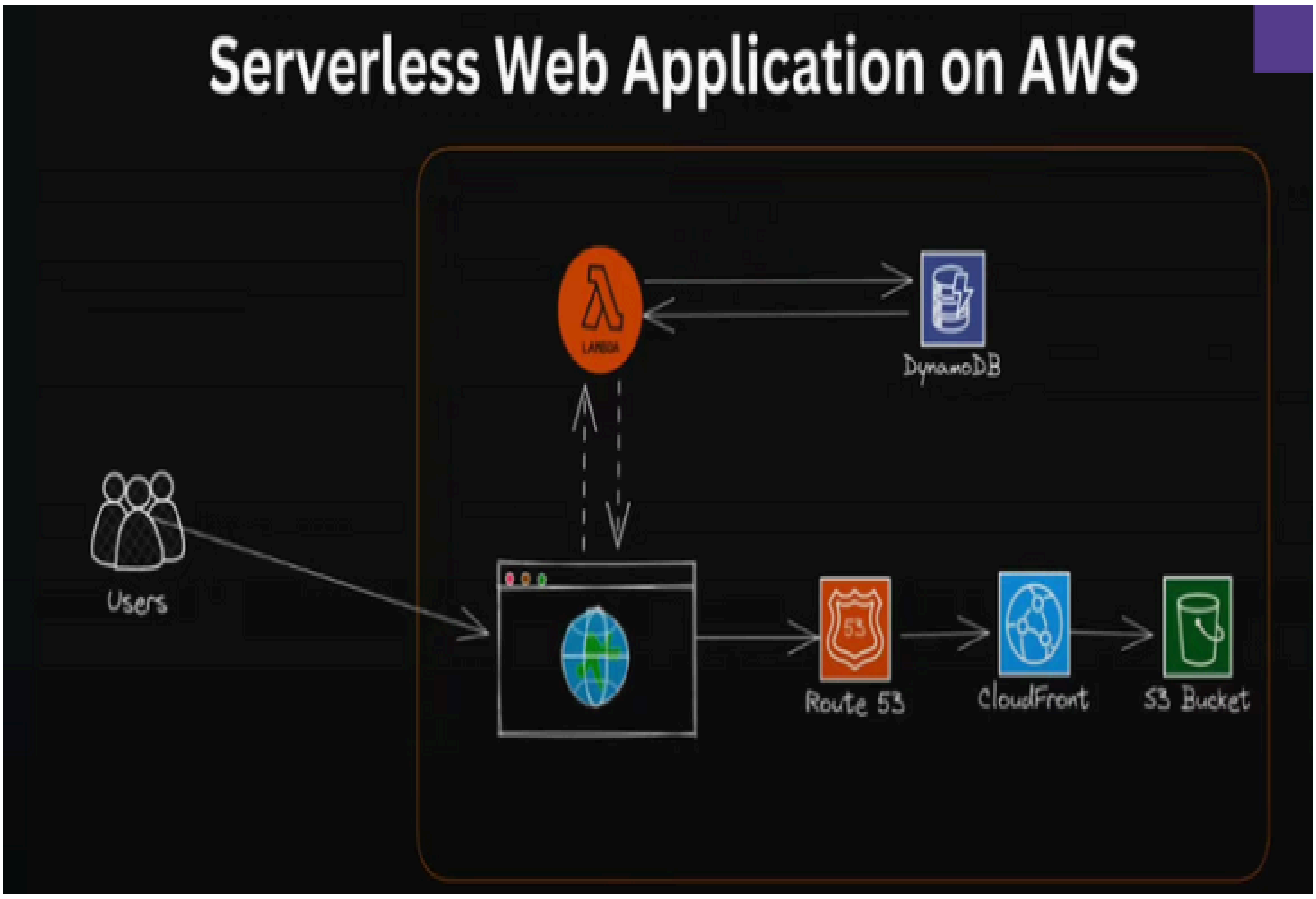
- We optimize queries to minimize DynamoDB read and write costs, leveraging features such as query filters, projections, and batch operations.
- By optimizing queries, we reduce latency and improve application responsiveness.

Scalability and Performance Considerations:

- We consider scalability and performance implications in our data design, ensuring that the database can handle growing data volumes and user loads.
- Provisioned throughput capacity is adjusted based on workload requirements, and auto-scaling features are utilized to adapt to changing demand.

By implementing these data design principles, we ensure that our serverless web application can efficiently manage data in DynamoDB, providing a reliable and scalable solution for developers and administrators.

ENTITY RELATIONSHIP DIAGRAM



5. SOFTWARE TESTING

5.1 OVERVIEW

Software testing is crucial to ensure the reliability, functionality, and performance of our serverless web application. We employ various testing methodologies and techniques throughout the development lifecycle to identify and rectify defects, ensuring a high-quality end product. Here's an overview of our software testing approach:

Unit Testing:

- We conduct unit tests for individual components of our application, including AWS Lambda functions, API endpoints, and data access layers.
- Unit tests are automated using testing frameworks such as Jest for JavaScript or Pytest for Python, ensuring thorough code coverage and early detection of bugs.

Integration Testing:

- Integration tests validate interactions between different components and external services, such as DynamoDB and Amazon S3.
- We use tools like AWS SAM (Serverless Application Model) for local testing and mocking AWS services, enabling comprehensive integration testing.

End-to-End Testing:

- End-to-end tests validate the entire application flow, simulating real user interactions and scenarios.

- We employ frameworks like Selenium or Cypress for web UI testing, ensuring that the application functions correctly from the user's perspective.

Load and Performance Testing:

- Load and performance tests assess the application's responsiveness and scalability under different workload conditions.
- We use tools like AWS LoadRunner or Apache JMeter to simulate concurrent user activity and measure system performance metrics.

Security Testing:

- Security testing is conducted to identify and mitigate potential vulnerabilities, ensuring the application adheres to best security practices.
- We employ static code analysis tools, vulnerability scanners, and penetration testing techniques to assess the application's security posture.

Regression Testing:

- Regression tests are executed after each code change or deployment to ensure that new features or modifications do not introduce unintended side effects.
- Automated regression test suites are run using continuous integration and deployment (CI/CD) pipelines, providing rapid feedback on code changes.

5.3 TEST CASES - A test case is a document with test data and expected outcomes for verifying compliance.

Test Case Title	**Description**	**Preconditions**	**Test Steps**	**Expected Result**	**Postconditions**
TC01	Create Item in DynamoDB Table	1. User authentication and authorization completed. 2.DynamoDB table exists and is accessible.	1. Navigate to "Create Item" page. 2. Input valid item details. 	1. Success message displayed. 2. Item added to DynamoDB with provided details. 	1. Item successfully created in DynamoDB. 2. Application remains stable. 3. Logs/notification generated.

			3. Click "Submit".	3. Item details shown on interface.	
TC02	Read Item from DynamoDB Table	1. User authentication and authorization completed. 2.DynamoDB table exists and contains items.	1. Navigate to "Read Item" page. 2. Select item to view details.	Item details displayed on interface.	Item details successfully retrieved from DynamoDB. Application remains stable.
TC03	Update Item in DynamoDB Table	1. User authentication and authorization completed. 2. DynamoDB table exists and contains items.	1. Navigate to "Update Item" page. 2. Select item to update. 3. Input new details. 	1. Success message displayed. 2. Item updated in DynamoDB with new details.	1. Item successfully updated in DynamoDB. 2. Application remains stable. 3. Logs/notification generated.

			4. Click "Update".		
TC04	Delete Item from DynamoDB Table	1. User authentication and authorization completed. 2. DynamoDB table exists and contains items.	1. Navigate to "Delete Item" page. 2. Select item to delete. 3. Confirm deletion.	1. Success message displayed. 2. Item removed from DynamoDB.	1. Item successfully deleted from DynamoDB. 2. Application remains stable. 3. Logs/notification generated.

6.REFERENCES

Technologies

JavaScript Complete Reference by Brendan Eich

HTML

www.w3school.com

www.javatpoint.com

CSS

www.w3school.com

by Bert Bos Research

Research

Amazon Web Services - https://aws.amazon.com