INTRODUCTION

1. INTRODUCTION

1.1 OVERVIEW

Building a serverless web application using AWS involves leveraging several key services to manage server infrastructure, scale automatically, and reduce operational complexity. AWS Lambda functions execute backend logic in response to HTTP requests routed through API Gateway, while Amazon S3 stores static assets, distributed globally via CloudFront for fast access. User authentication and authorization are handled by AWS Cognito, ensuring secure access to application resources. Data storage is managed by Amazon DynamoDB, a fully managed NoSQL database. AWS Amplify simplifies front-end integration and deployment, enhancing development speed. This serverless architecture provides scalability, cost efficiency, security, and reduced operational overhead, allowing developers to focus on building and delivering application features.

LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 EXISTING SYSTEM

An existing system in a serverless web application using AWS typically includes several integrated AWS services that work together to provide a fully functional and scalable application environment. Here's an overview of these components:

1. **Amazon S3**: This service is used to store static assets such as HTML, CSS, JavaScript files, images, and other media. S3 ensures high durability, scalability, and availability of these static resources.
2. **Amazon CloudFront**: CloudFront is a content delivery network (CDN) that caches and delivers static and dynamic content globally with low latency. It integrates seamlessly with S3, distributing content efficiently to users worldwide.
3. **AWS API Gateway**: API Gateway serves as the entry point for all API requests. It routes HTTP requests to appropriate AWS Lambda functions, managing request routing, security, and throttling.
4. **AWS Lambda**: Lambda functions handle the execution of backend logic. They are triggered by API Gateway requests or other AWS service events, running the necessary code without managing servers. Lambda supports multiple programming languages and scales automatically with demand.
5. **Amazon DynamoDB**: This fully managed NoSQL database is used for storing and retrieving application data. DynamoDB offers high performance, scalability, and flexibility for handling various data models.
6. **AWS Cognito**: Cognito manages user authentication and authorization, providing user sign-up, sign-in, and access control. It supports integration with social identity providers and ensures secure handling of user data.
7. **AWS Amplify**: Amplify is a development platform that simplifies building and deploying mobile and web applications. It offers a CLI, libraries, and UI components that integrate easily with AWS services, streamlining the development process.
8. **Amazon CloudWatch**: CloudWatch monitors application performance, logs, and operational health. It provides insights and alerts based on predefined metrics, helping to ensure the application runs smoothly.

2.2 PROPOSED SYSTEM

The proposed system for a serverless web application using AWS aims to enhance scalability, reduce operational complexity, and improve cost efficiency. This system leverages a range of AWS services to provide a robust and secure application environment. Here's a detailed outline of the proposed system:

1. **Frontend Hosting and Delivery**:
   - **Amazon S3**: Store static files such as HTML, CSS, JavaScript, images, and videos. S3 provides high durability and availability for these assets.
   - **Amazon CloudFront**: Use CloudFront as a content delivery network (CDN) to distribute static content globally, ensuring low latency and high transfer speeds for users around the world.
2. **Backend Logic and API Management**:
   - **AWS API Gateway**: Serve as the primary interface for HTTP requests. API Gateway will manage routing, security, throttling, and API versioning, ensuring reliable and scalable API management.
   - **AWS Lambda**: Execute backend code in response to API Gateway requests. Lambda functions will handle business logic, data processing, and integration with other AWS services, scaling automatically based on demand.
3. **Data Storage and Management**:
   - **Amazon DynamoDB**: Provide a fully managed NoSQL database service for storing application data. DynamoDB will ensure fast and flexible querying capabilities with automatic scaling and high availability.
   - **Amazon RDS (Relational Database Service)**: For applications requiring relational database capabilities, RDS will be used to manage and scale MySQL, PostgreSQL, or other SQL databases efficiently.
4. **User Authentication and Authorization**:
   - **AWS Cognito**: Manage user authentication, sign-up, sign-in, and access control. Cognito will support integration with social identity providers and secure user data management.

5. **Application Development and Deployment**:

- **AWS Amplify**: Simplify front-end development and deployment. Amplify will provide CLI tools, libraries, and UI components for seamless integration with AWS services, accelerating the development process.

2.3 TECHNOLOGIES USED

2.3.1 Amazon S3 (Simple Storage Service)

- **Purpose**: Storage for static assets such as HTML, CSS, JavaScript files, images, and videos.
- **Features**:
  - **Durability and Availability**: Provides 99.999999999% durability and 99.99% availability of objects over a given year.
  - **Scalability**: Automatically scales storage capacity as needed without any management required.
  - **Security**: Supports encryption for data at rest and in transit, fine-grained access control policies.
  - **Static Website Hosting**: Can host static websites directly from an S3 bucket.

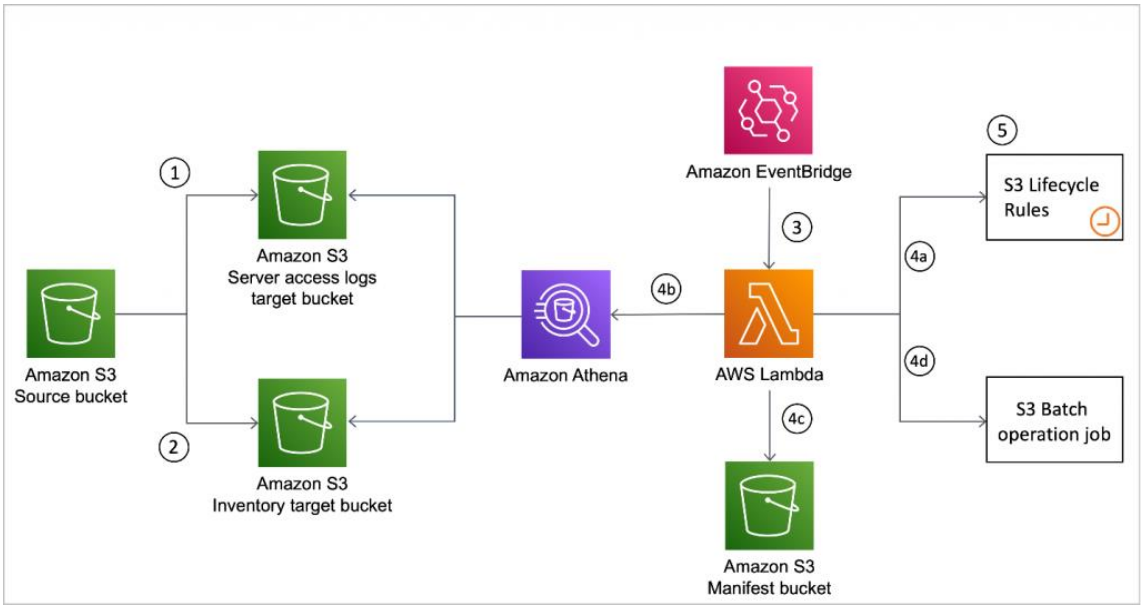


Fig: Amazon S3(Simple Storage Service**)**

2.3.2 Amazon CloudFront

Amazon CloudFront is a content delivery network (CDN) service offered by Amazon Web Services (AWS) that delivers web content, including both static and dynamic files, to users with low latency and high transfer speeds. CloudFront helps improve the performance, reliability, and security of web applications by caching content at strategically placed edge locations worldwide.

- **Purpose**: Global content delivery network (CDN) to deliver static and dynamic content with low latency.
- **Features**:
  - **Global Distribution**: Distributes content through a network of edge locations worldwide.
  - **Performance**: Reduces latency by caching content at edge locations closer to users.
  - **Security**: Provides DDoS protection, SSL/TLS encryption, and integrates with AWS WAF for additional security.
  - **Customizable**: Supports custom headers and URL rewrites for dynamic content delivery.

2.3.3. AWS Lambda

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS) that allows you to run code without provisioning or managing servers. Lambda automatically scales your application by running code in response to events, and you only pay for the compute time you consume, which ensures cost efficiency.

- **Purpose**: Serverless compute service that runs backend code in response to events.
- **Features**:
  - **Automatic Scaling**: Automatically scales based on the number of incoming requests.
  - **Cost Efficiency**: Pay only for the compute time consumed; no charge when the code is not running.
  - **Integration**: Easily integrates with other AWS services like DynamoDB, S3, and API Gateway.
  - **Multiple Languages**: Supports various programming languages including Python, Node.js, Java, Go, and Ruby.

2.3.4. AWS API Gateway

- **Purpose**: Fully managed service to create, publish, maintain, monitor, and secure APIs.
- **Features**:
  - **Request Handling**: Routes incoming requests to appropriate backend services (e.g., Lambda functions, HTTP endpoints).
  - **Security**: Supports authorization and access control using AWS IAM and AWS Cognito.
  - **Monitoring**: Provides detailed monitoring and logging through CloudWatch.
  - **Throttling and Quotas**: Manages traffic and applies rate limits to prevent abuse.

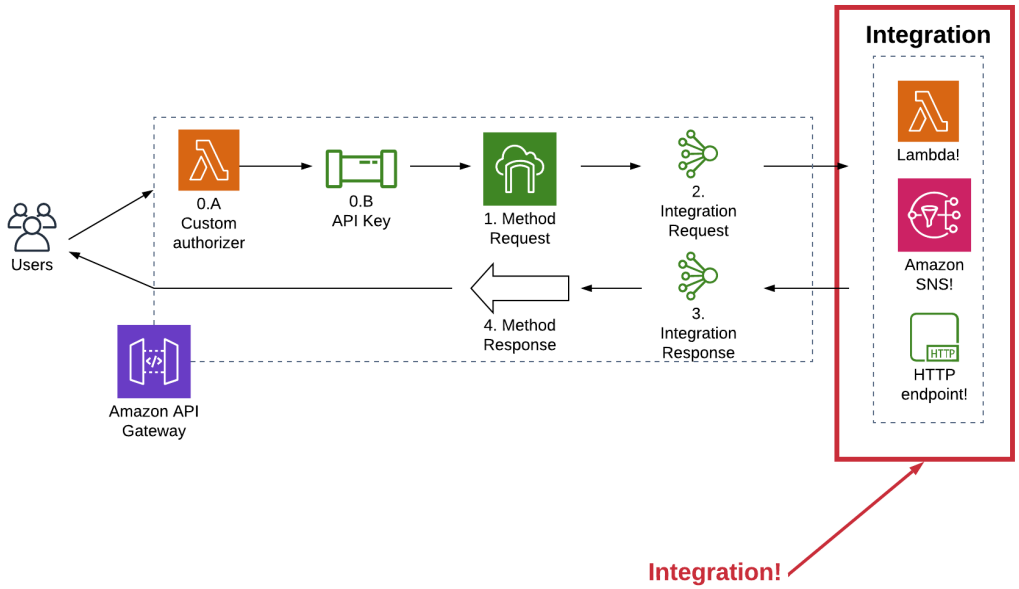


Fig: AWS API Gateway

2.3.5Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service provided by Amazon Web Services (AWS) that offers fast and predictable performance with seamless scalability. DynamoDB allows developers to store and retrieve any amount of data and serves as a highly available and fault-tolerant database solution. It is designed to handle high request rates and large amounts of data, making it suitable for applications that require consistent, low-latency data access.

- **Purpose**: Fully managed NoSQL database service for storing and retrieving application data.
- **Features**:
  - **Performance**: Offers single-digit millisecond response times at any scale.
  - **Scalability**: Automatically scales read and write capacity based on application traffic.
  - **High Availability**: Data is automatically replicated across multiple AWS regions for high availability and durability.
  - **Flexible Data Models**: Supports key-value and document data models.

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

The analysis model is a concise, precise abstraction of what the desired system must do, and not how it will be done after the study of the existing system is completed. This basically includes system study and the requirement analysis. Interacting with the clients regarding their requirements and expectations from the system does requirement analysis.

The steps, which are essential for system analysis, are:

1. Research and define essential components
2. Analyze current processes and identify gaps.
3. Interview users, Trainee, Trainers and other concerned personnel regarding essential components and current processes.
4. Write requirements document.
5. Define standards for standards, policies, and procedures.
6. Review draft requirements document with users, Trainee, Trainers and other concerned personnel.
7. Update and expand project plan.

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose. Analysis specifies what the system should do.

HARDWARE AND SOFTWARE TECHNOLOGY USED

4. HARDWARE AND SOFTWARE TECHNOLOGY USED

4.1 HARDWARE REQUIREMENTS

PROCESSOR	:	Pentium – iv and above
RAM	:	256 Mb
HARD DISK	:	4.3 Gb
FLOPPY DISK	:	1.44 Mb
MONITOR	:	15” Color Monitor
KEYBOARD	:	Standard 102 keys
MOUSE	:	3 Buttons

4.2 SOFTWARE REQUIREMENTS

OPERATING SYSTEM	:	Windows XP
FRONT END	:	HTML , CSS , JavaScript
BACK END	:	Python, MySQL ,Dynamo DB
WEB SERVER	:	AWS

SYSTEM DESIGN

5. SYSTEM DESIGN

5.1 ENTITY RELATIONSHIP DIAGRAM

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique. The purpose of ER Diagram is to represent the entity framework infrastructure.

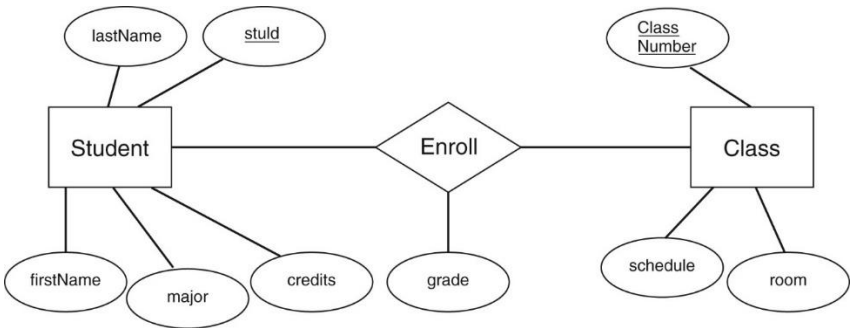


Fig: E-R Diagram for Student Information

5.2 USE CASE DAIGRAM

Use-case diagrams illustrate and define the context and requirements of either an entire system or the important parts of the system. You can model a complex system with a single use-case diagram, or create many use-case diagrams to model the components of the system. You would typically develop use-case diagrams in the early phases of a project and refer to them throughout the development process.

Use-case diagrams are helpful in the following situations:

- Before starting a project, you can create use-case diagrams to model a business so that all participants in the project share an understanding of the workers, customers, and activities of the business.
- While gathering requirements, you can create use-case diagrams to capture the system requirements and to present to others what the system should do.
- During the analysis and design phases, you can use the use cases and actors from your use-case diagrams to identify the classes that the system requires.
- During the testing phase, you can use use-case diagrams to identify tests for the system.

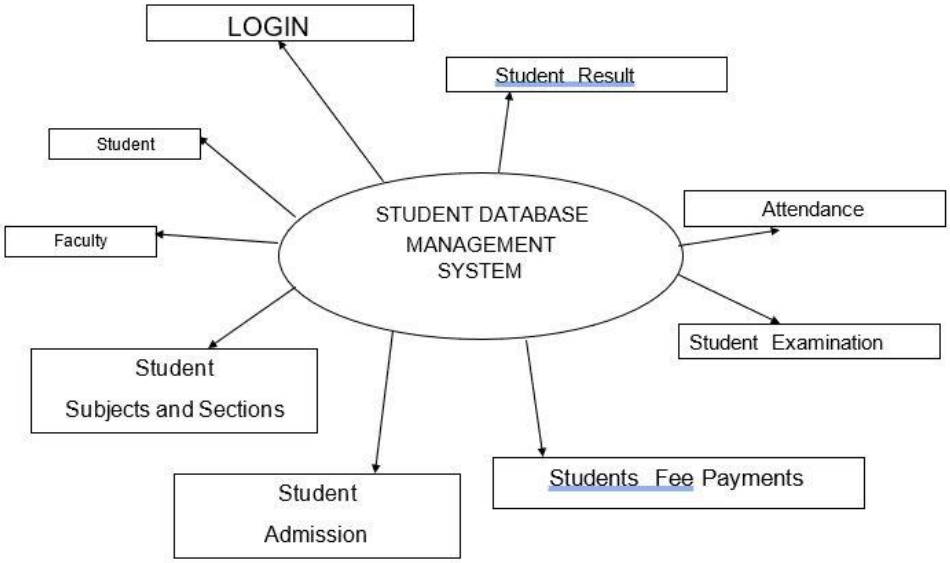


Fig. Dataflow Diagram

SYSTEM IMPLEMENTATION

6.SYSTEM IMPLEMENTATION

The purpose of System Implementation can be summarized as follows: making the new system available to a prepared set of users (the deployment), and positioning on-going support and maintenance of the system within the Performing Organization (the transition). At a finer level of detail, deploying the system consists of executing all steps necessary to educate the Consumers on the use of the new system, placing the newly developed system into production, confirming that all data required at the start of operations is available and accurate, and validating that business functions that interact with the system are functioning properly. Transitioning the system support responsibilities involves changing from a system development to a system support and maintenance mode of operation, with ownership of the new system moving from the Project Team to the Performing Organization. A key difference between System Implementation and all other phases of the lifecycle is that all project activities up to this point have been performed in safe, protected, and secure environments, where project issues that arise have little or no impact on day-to-day business operations. Once the system goes live, however, this is no longer the case. Any miscues at this point will almost certainly translate into direct operational and/or financial impacts on the Performing Organization. It is through the careful planning, execution, and management of System Implementation activities that the Project Team can minimize the likelihood of these occurrences, and determine appropriate contingency plans in the event of a problem.

6.1 CODING

1. index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Data</title>
    <style>
        body {
            background-color: #f0f0f0; /* Light gray background */
            color: #333; /* Dark gray text */
            font-family: Arial, sans-serif; /* Use Arial font */
        }

        h1 {
            color: #007bff; /* Blue heading text */
        }

        .container {
            max-width: 600px; /* Limit width to 600px */
            margin: 0 auto; /* Center the container */
            padding: 20px; /* Add padding */
            background-color: #fff; /* White background */
            border-radius: 10px; /* Rounded corners */
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Add shadow */
        }
```

```
input[type="text"], input[type="submit"] {
        width: 100%;
        padding: 10px;

        box-sizing: border-box;
        border: 1px solid #ccc;
        border-radius: 5px;
    }

    input[type="submit"] {
        background-color: #007bff; /* Blue submit button */
        color: #fff; /* White text */
        cursor: pointer; /* Add pointer cursor */
    }

    input[type="submit"]:hover {
        background-color: #0056b3; /* Darker blue on hover */
    }

    table {
        width: 100%;

  border-collapse: collapse;
    }

    th, td {
        padding: 8px;
        text-align: left;
        border-bottom: 1px solid #ddd;
    }

    th {
```

```
background-color: #f2f2f2; /* Light gray header background */
        }


    </style>
</head>
<body>
    <div class="container">
        <h1>Save and View Student Data</h1>
        <label for="studentid">Student ID:</label><br>
        <input type="text" name="studentid" id="studentid"><br>

        <label for="name">Name:</label><br>
        <input type="text" name="name" id="name"><br>

        <label for="class">Class:</label><br>
        <input type="text" name="class" id="class"><br>

        <label for="age">Age:</label><br>
        <input type="text" name="age" id="age"><br>

        <br>
        <input type="submit" id="savestudent" value="Save Student Data">
        <p id="studentSaved"></p>

        <br>
        <input type="submit" id="getstudents" value="View all Students">
        <br><br>
        <div id="showStudents">
            <table id="studentTable">
                <colgroup>
```

```
<col style="width:20%">
                <col style="width:20%">
                <col style="width:20%">




                <col style="width:20%">
            </colgroup>
            <thead>
                <tr>
                    <th>Student ID</th>
                    <th>Name</th>
                    <th>Class</th>
                    <th>Age</th>
                </tr>
            </thead>
            <tbody>
                <!-- Student data will be displayed here -->
            </tbody>


        </table>
    </div>
</div>

<script src="scripts.js"></script>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.0/jquery.min.js"></script>
</body>
</html>
```

2. Script.js:

```
// Add your API endpoint here
var API_ENDPOINT = "https://sasftli1ng.execute-api.ap-south-1.amazonaws.com/prod";



// AJAX POST request to save student data
document.getElementById("savestudent").onclick = function(){
    var inputData = {
        "studentid": $('#studentid').val(),
        "name": $('#name').val(),
        "class": $('#class').val(),
        "age": $('#age').val()
    };
    $.ajax({
        url: API_ENDPOINT,
        type: 'POST',
        data:  JSON.stringify(inputData),
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            document.getElementById("studentSaved").innerHTML = "Student Data Saved!";
        },
        error: function () {
            alert("Error saving student data.");
        }
    });
}

// AJAX GET request to retrieve all students
document.getElementById("getstudents").onclick = function(){
    $.ajax({
        url: API_ENDPOINT,
```

```
        type: 'GET',
            contentType: 'application/json; charset=utf-8',
            success: function (response) {
                $('#studentTable tr').slice(1).remove();
                jQuery.each(response, function(i, data) {
                    $("#studentTable").append("<tr> \
        >" + data['studentid'] + "</td> \

                        <td>" + data['class'] + "</td> \
                        <td>" + data['age'] + "</td> \
                        </tr>");
                });
            },
            error: function () {
                alert("Error retrieving student data."



            }
        });
```

3.getstudentdata.py

```
import json
 import boto3

def lambda_handler(event, context):
  # Initialize a DynamoDB resource object for the specified region
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')

# Select the DynamoDB table named 'studentData'
  table = dynamodb.Table('studentData')
```

```
    # Scan the table to retrieve all items
    response = table.scan()
    data = response['Items']

    # If there are more items to scan, continue scanning until all items are
    while 'LastEvaluatedKey' in response:
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])

    # Return the retrieved data
    return data
```

4. inserystudentdata.py

```
import json
import boto3

# Create a DynamoDB object using the AWS SDK
dynamodb = boto3.resource('dynamodb')
# Use the DynamoDB object to select our table
table = dynamodb.Table('studentData')

# Define the handler function that the Lambda service will use as an
def lambda_handler(event, context):
    # Extract values from the event object we got from the Lambda service
    student_id = event['studentid']
    name = event['name']
```

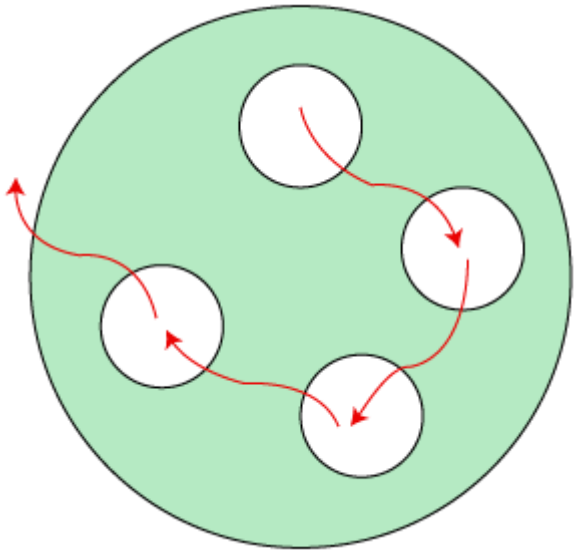
```
    # Write student data to the DynamoDB table and save the response
   response = table.put_item(
      Item={


'name': name,
'class': student_class,
'age': age
  }
     )


    # Return a properly formatted JSON object
  return {
    'statusCode': 200,
      'body': json.dumps('Student data saved successfully!')
    }
```

SYSTEM TESTING

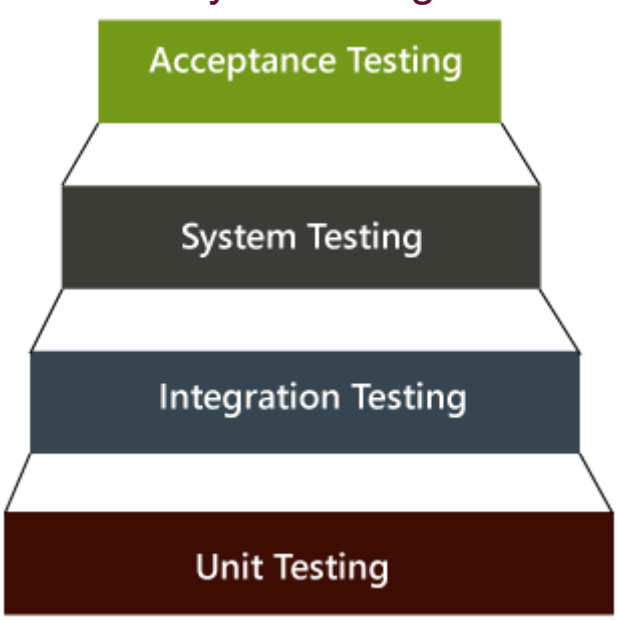
7. SYSTEM TESTING

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as **System testing**. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system. It is **end-to-end testing** where the testing environment is similar to the production environment.



There are four levels of software testing: unit testing, integration testing, system testing and acceptance testing, all are used for the testing purpose. Unit Testing used to test a single software; Integration Testing used to test a group of units of software, System Testing used to test a whole system and Acceptance Testing used to test the acceptability of business requirements. Here we are discussing system testing which is the third level of testing levels.



There are mainly two widely used methods for software testing, one is **White box testing** which uses internal coding to design test cases and another is black box testing which uses GUI or user perspective to develop test cases.

System testing falls under Black box testing as it includes testing of the external working of the software.

Testing follows user's perspective to identify minor defects.

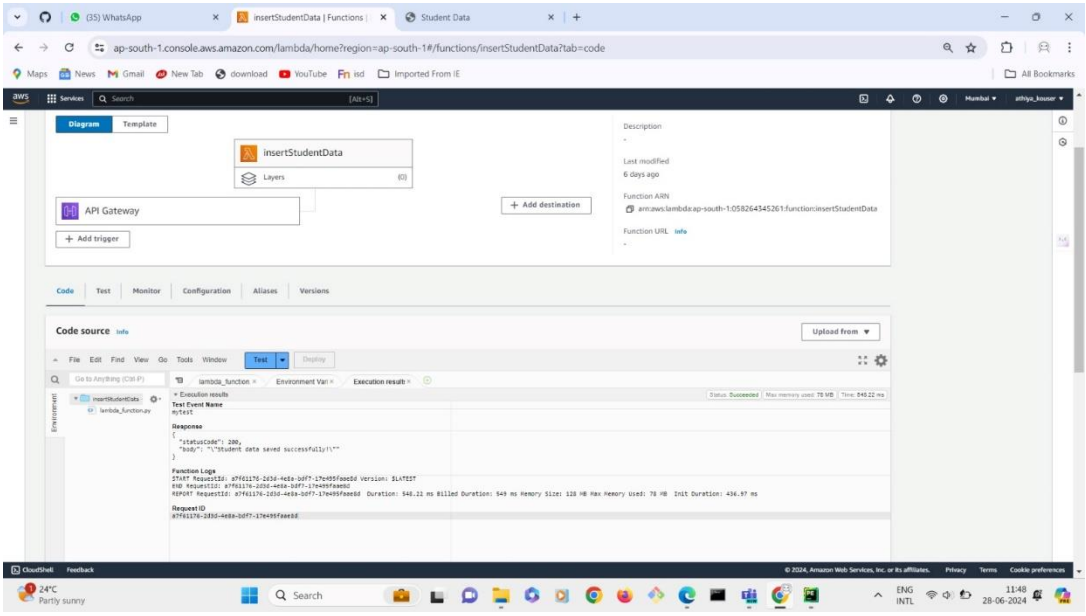
System Testing includes the following steps.

- Verification of input functions of the application to test whether it is producing the expected output or not.
- Testing of integrated software by including external peripherals to check the interaction of various components with each other.
- Testing of the whole system for End to End testing.
- Behavior testing of the application via auser's experience.

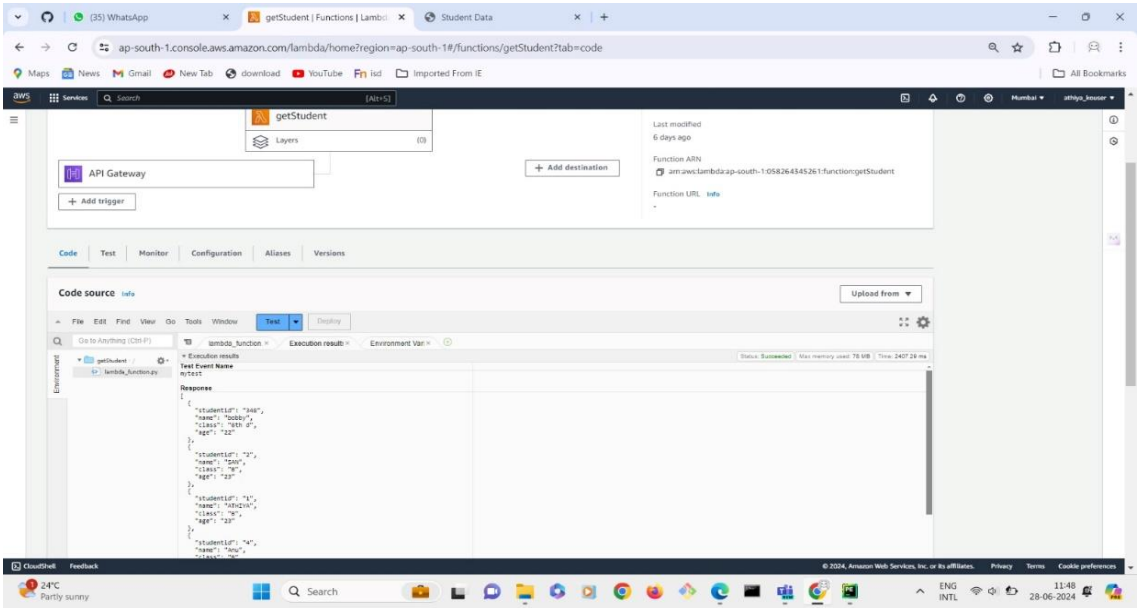
RESULTS AND SNAPSHOTS

8. RESULTS AND SNAPSHOTS

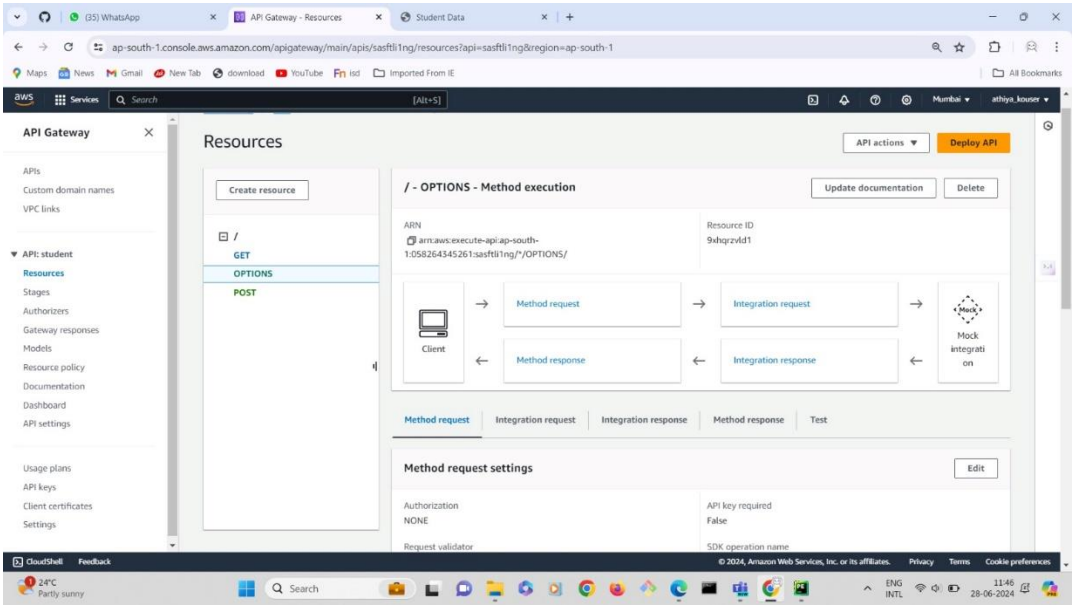
1.Testing post method:



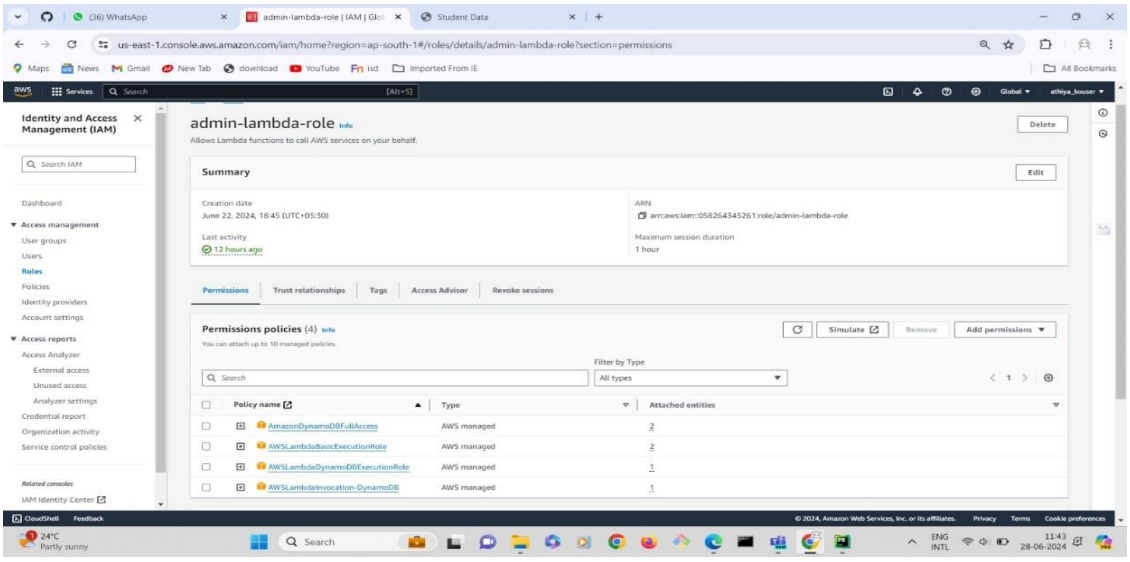
2.Testing get method:



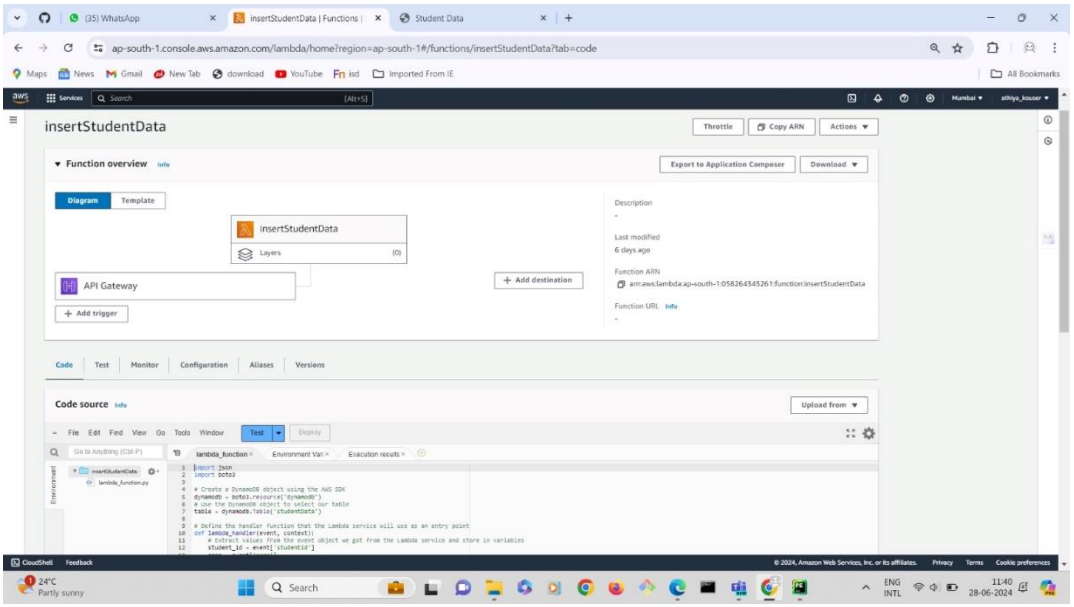
3. Method Execution and cors policy:



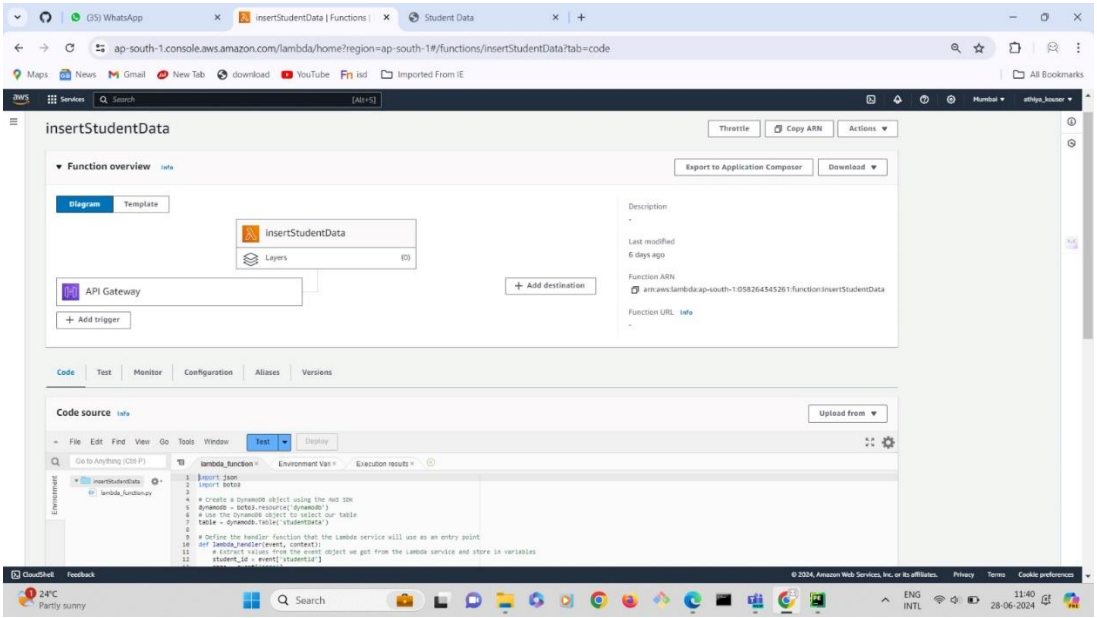
5. IAM roles assigned:



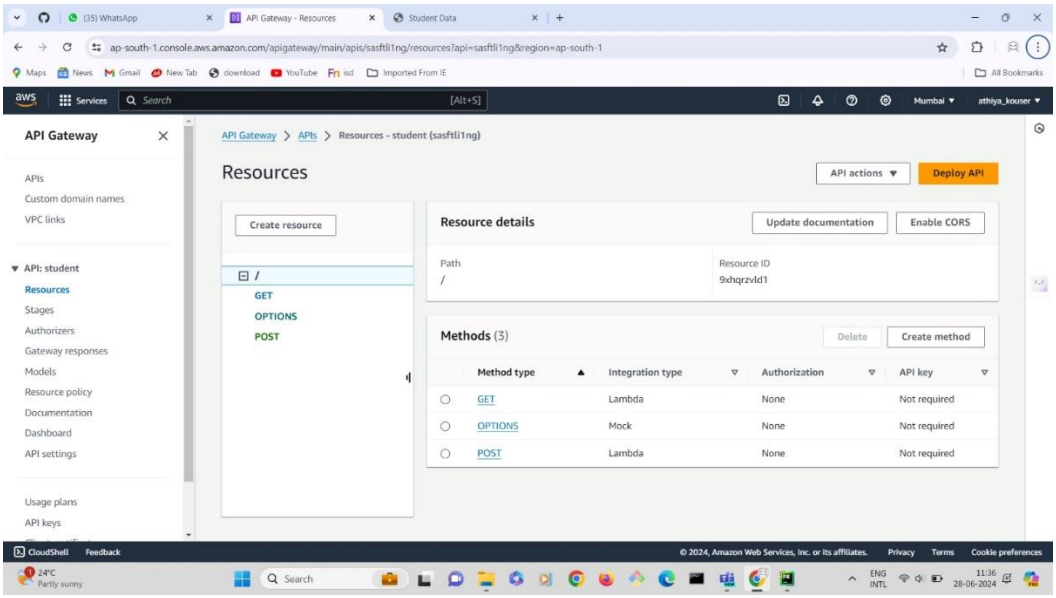
6. Lambda function for POST method:



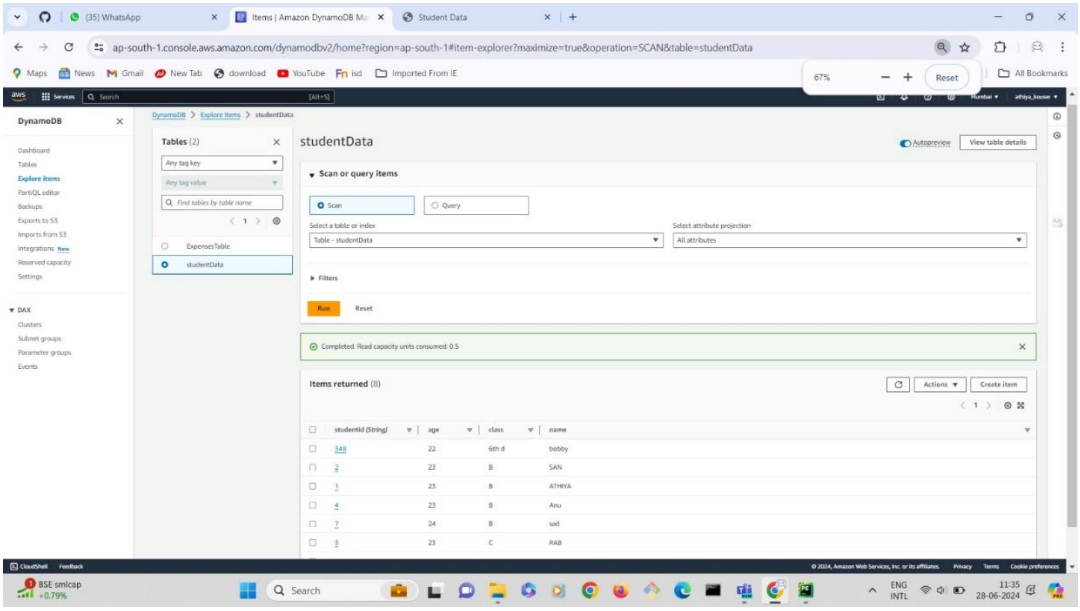
7. Lambda function for Get method:



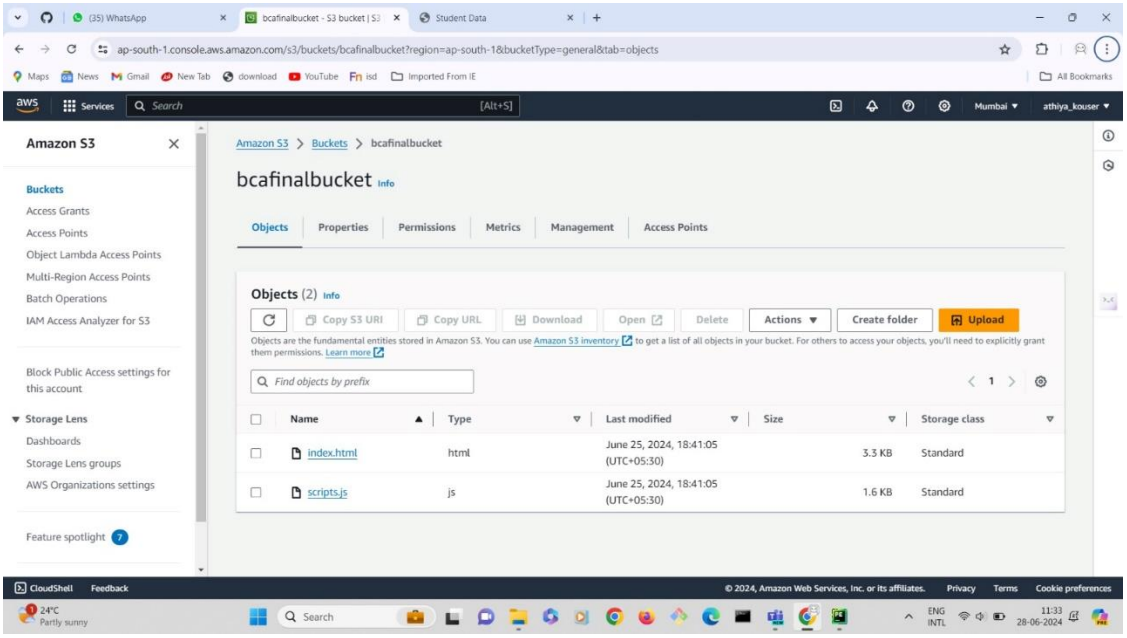
8. API fetched between Lambda function and s3 bucket:



9. value stored in DynamoDB:



10. code stored in s3 bucket:



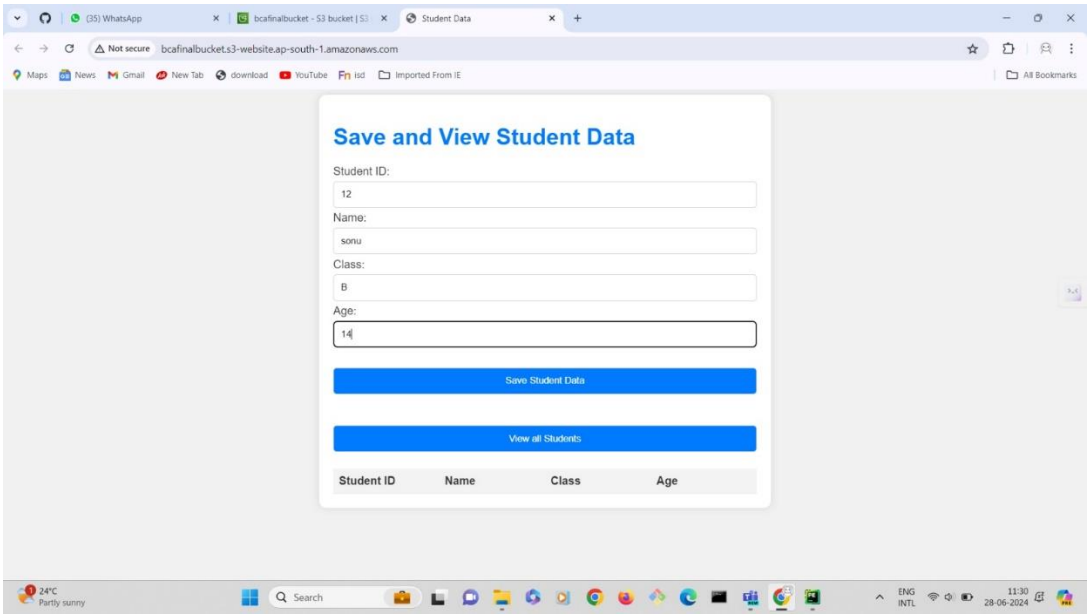


Fig: Insert Student-Data

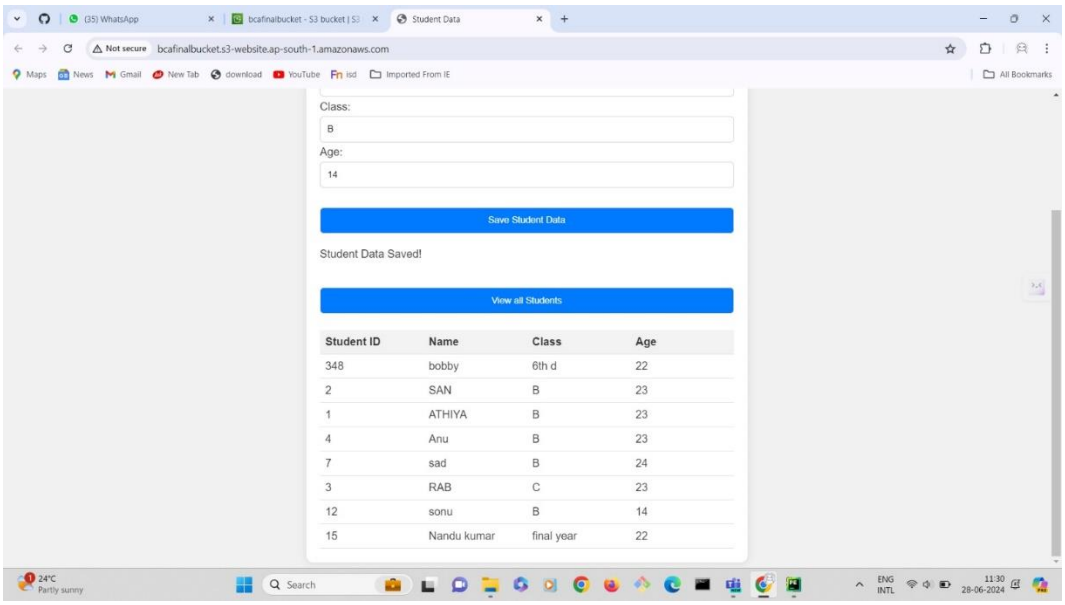


Fig: View Student-Data

CONCLUSION AND FUTURE SCOPE

9. CONCLUSION

Serverless computing has revolutionized web applications are developed and deployed. By leveraging AWS (Amazon Web Services), developers can build scalable, cost-effective, and resilient web applications without the need to manage server infrastructure.

Using AWS for serverless web applications provides numerous advantages, including cost savings, ease of management, and enhanced scalability. By abstracting away the underlying infrastructure, AWS enables developers to focus on innovation and delivering value to end-users. While there are some considerations to be mindful of, the benefits of serverless architectures often outweigh the challenges. Organizations can leverage AWS's comprehensive suite of services to build robust, scalable, and secure web applications, driving business growth and operational efficiency. As serverless technology continues to evolve, it is poised to become an integral part of modern web application development.

9.1 FUTURE SCOPE

The future of serverless web applications using AWS is promising, marked by continuous advancements in performance, security, and developer experience. AWS is expected to enhance serverless performance by reducing cold start times and expanding edge computing capabilities, bringing computation closer to users and reducing latency. The integration of AI and machine learning services, such as AWS SageMaker, will enable serverless applications to leverage advanced analytics and automation. New AWS services and tools will simplify development, improve resource management, and facilitate cross-cloud deployments. Enhanced security features and compliance automation will ensure robust protection and adherence to industry standards. As serverless architectures promote energy efficiency, they contribute to sustainability goals. Increased adoption across diverse industries, driven by the need for scalable and cost-effective solutions, will see serverless architectures powering real-time applications, IoT, and interactive web and mobile experiences. Overall, AWS's evolving serverless ecosystem will enable organizations to innovate rapidly, achieve cost savings, and maintain agility in the digital age.

REFERENCES

WEBSITES

- **www.w3schools.com**
- **www.php.net**
- **www.mysql.com**
- **www.wikipedia**
- **www.stackoverflow.com**
- **www.tutorialpoint.com**