

# **SAAS APPLICATION ON AWS**

**A NAAN MUDHALVAN PROJECT REPORT**

**SUBMITTED BY**

**BALAJI M                      912421106003**

**BALAKUMAR B              912421106004**

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**



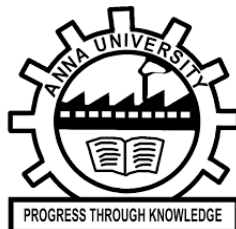
**SHANMUGANATHAN ENGINEERING COLLEGE**

**ARASAMPATTI, PUDUKKOTTAI – 622 507**

**YEAR & SEMESTER    : IV & VII**

**SUBJECT CODE            : NM1050**

**COURSE NAME            : SaaS**



**ANNA UNIVERSITY::CHENNAI 600 025**

**NOV/DEC 2024**

**ANNA UNIVERSITY::CHENNAI 600 025**

**BONAFIDE CERTIFICATE**

Certified that this Naan Mudhalvan project report “**SAAS APPLICATION ON AWS**” is the bonafide work of “**BALAJI M (912421106003), BALAKUMAR B (912421106004)** ” who carried out the project work under my guidance.

**SIGNATURE**

**Mrs. D. LATHA, M.E.,**

**NAAN MUDHALAVAN COORDINATOR**

**Assistant Professor,**

Dept. of Electronics and Comm. Engg,

Shanmuganathan Engineering College,

Arasampatti-622 507

**SIGNATURE**

**Dr. A.MUTHU MANICKAM , M.E., Ph.D.,**

**HEAD OF THE DEPARTMENT**

**ASSISTANT PROFESSOR,**

Department of Electronics & Communication  
Engineering,

Shanmuganathan Engineering College,

Arasampatti – 622 507

---

Submitted for the Internship viva-voice on \_\_\_\_\_

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## ACKNOWLEDGMENT

At this pleasing moment having successfully completed our internship report, we wish to convey our sincere thanks to our beloved chairperson **Mrs. PICHAPPA VALLIAMMAL**, correspondent **Dr. P. MANIKANDAN B.E**, director (Academic) Shri **M.SHANMUGANATHAN**, director(Administration) Shri **M. PICHAPPA** and honourable secretary **Mr. M. VISWANATHAN** for their extensive support.

I thankful to our principal **Dr. KL. MUTHURAMU M.E(W.R)., M.E(S.E)., Ph.D., FIE., M.I.S.T.E.**, Shanmuganathan engineering college, for providing the opportunity to conduct our project.

I extend our gratitude to **Dr. A.MUTHU MANICKAM M.E., Ph.D.**, the head of the department and our internship co-ordinator of Electronics and communication engineering for providing a valuable suggestion and supports given through the study.

Gratitude never fails. We are grateful to our dynamic and effective internal guide **Asst. Prof. Mrs. D. LATHA, M.E, AP/ECE**. for his valuable innovative suggestion, constructive interactions, constant encouragement and valuable helps that have been provided us throughout the project.

I also express my heartfelt thanks to all other staff members of Electronics communication engineering department for their support. Above all, we thank our parents, for affording us the valuable education till now.

# **ABSTRACT**

The deployment of Software-as-a-Service (SaaS) applications on Amazon Web Services (AWS) offers a scalable, reliable, and cost-efficient solution for providing cloud-based services. This abstract explores the process of deploying a SaaS application on AWS, focusing on best practices, key AWS services, and architectural considerations. A SaaS application typically involves multi-tenant architecture, where a single instance of the software serves multiple customers, or tenants, each with isolated data and configurations. AWS provides various services that support the deployment, scalability, and security of SaaS applications, such as Amazon Elastic Compute Cloud (EC2), Amazon S3 for storage, Amazon RDS for database management, and Amazon Elastic Load Balancer (ELB) for distributing traffic. Additionally, AWS offers tools like AWS Lambda for serverless computing, AWS Auto Scaling for dynamic resource scaling, and Amazon CloudFront for content delivery. Key challenges in SaaS deployment include ensuring high availability, managing security, implementing proper multi-tenancy isolation, and handling scaling requirements. Solutions like AWS Identity and Access Management (IAM) and Virtual Private Cloud (VPC) can help address these concerns. Monitoring and logging are critical for maintaining the health of the application, with services like AWS CloudWatch and AWS CloudTrail providing insights into performance and security.

## TABLE OF CONTENT

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	I
	LIST OF FIGURES	III
1	<b>INTRODUCTION</b>	1
	1.1    SAAS APPLICATION ON AWS	1
2	<b>SYSTEM ARCHITECTURE</b>	2
3	<b>FRONTEND LAYER</b>	3
	3.1    HTML & CSS	3
	3.2    JAVASCRIPT	5
4	<b>AWS SERVICE SELECTION</b>	8
	4.1    NODE.JS	8
5	<b>INTERFACE OF THE PROJECT</b>	11
6	<b>CONCLUSION</b>	13

## LIST OF FIGURE

FIGURE NO.	NAME OF THE FIGURE	PAGE NO
2.1	System Architecture	2
3.1.1	HTML	3
3.1.2	HTML Code Execution	4
3.1.3	CSS	4
3.1.4	CSS Code Execution	5
3.2.1	JavaScript	6
3.2.2	JS Code Execution	7
4.1.1	Node.js	8
4.1.2	node JS Code Execution	9
4.2.1	STARTING INTERFACE	10
4.2.2	INPUT LISTED INTERFACE	10
5.1	AWS OUTPUT STORED INTERFACE	12

# CHAPTER 1

## INTRODUCTION

### 1.1 SAAS APPLICATION ON AWS :

In today's digital age, **Software as a Service (SaaS)** has emerged as one of the most popular delivery models for software applications. SaaS allows businesses to offer software solutions over the internet on a subscription basis, enabling users to access features and functionalities from anywhere without the need to install or maintain the software themselves. This model provides a range of benefits, including lower upfront costs, ease of updates, and scalability. In the modern software landscape, **Software as a Service (SaaS)** has become a widely adopted model for delivering software solutions over the internet. SaaS applications are hosted in the cloud and made available to users on a subscription basis. This model offers numerous benefits, including lower costs, scalability, ease of access, and reduced maintenance overhead for both service providers and customers.

**Amazon Web Services (AWS)** is a leading cloud platform that provides a robust and scalable infrastructure, making it an ideal choice for deploying SaaS applications. AWS offers a wide range of services that enable developers to build, deploy, and manage applications with high availability, security, and performance. By leveraging AWS, organizations can focus on delivering value to their customers while offloading the complexities of infrastructure management. This guide aims to walk you through the process of deploying a SaaS application on AWS, covering all the essential components needed to build a reliable and scalable cloud-based solution. Whether you are starting a new project or migrating an existing application to the cloud, this guide will help you leverage AWS services to maximize efficiency, performance, and security.

## CHAPTER 2

# SYSTEM ARCHITECTURE

Designing a robust system architecture is crucial for the success of any SaaS application. Leveraging AWS's extensive set of services enables you to build a scalable, resilient, and secure solution that can handle varying workloads and customer demands. This section outlines a **multi-tier architecture** for deploying a SaaS application on AWS, covering all essential components from the front end to the backend and database layers.

The frontend serves as the user interface, allowing different types of users (administrators, faculty, and students) to interact with the system. HTML provides the structure of the web pages, CSS styles them, and JavaScript ensures interactivity, allowing for real-time data validation, dynamic page updates, and interactive elements.

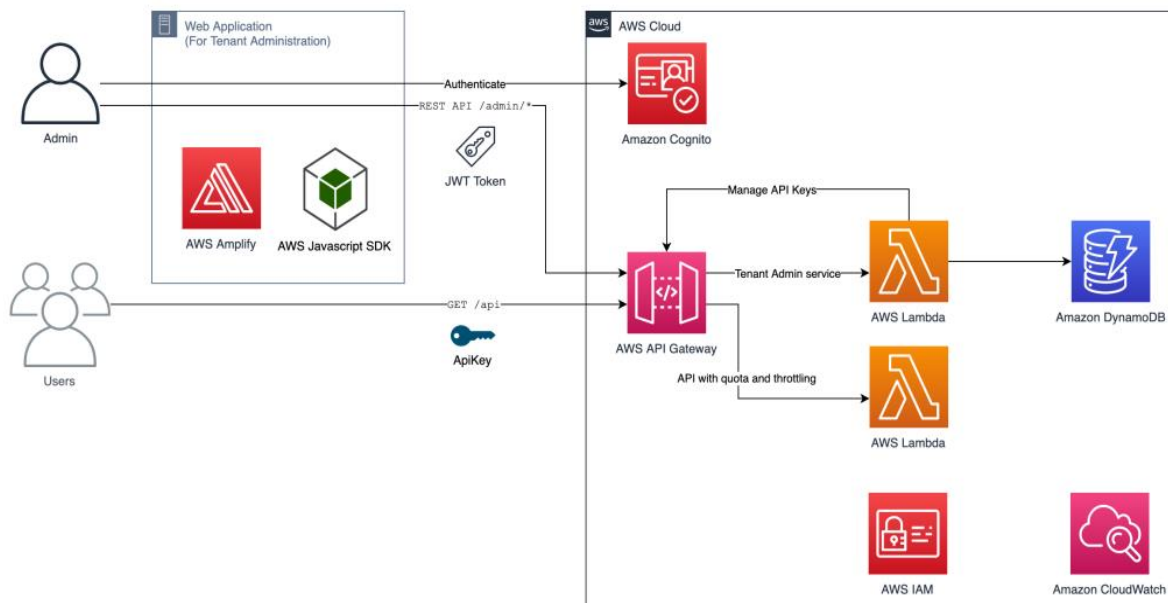


Fig 2.1 System Architecture

The backend is responsible for processing requests from the frontend, performing business logic, and handling interactions with the PostgreSQL database. It serves as an intermediary between the database and the user interface, managing queries, updates, and user permissions.



## **CHAPTER-3**

### **FRONTEND LAYER**

The frontend is the client-facing component of the architecture, responsible for interacting with users and displaying inventory data. Built using HTML, CSS, and JavaScript, this layer is designed for responsiveness, usability, and efficient data handling.

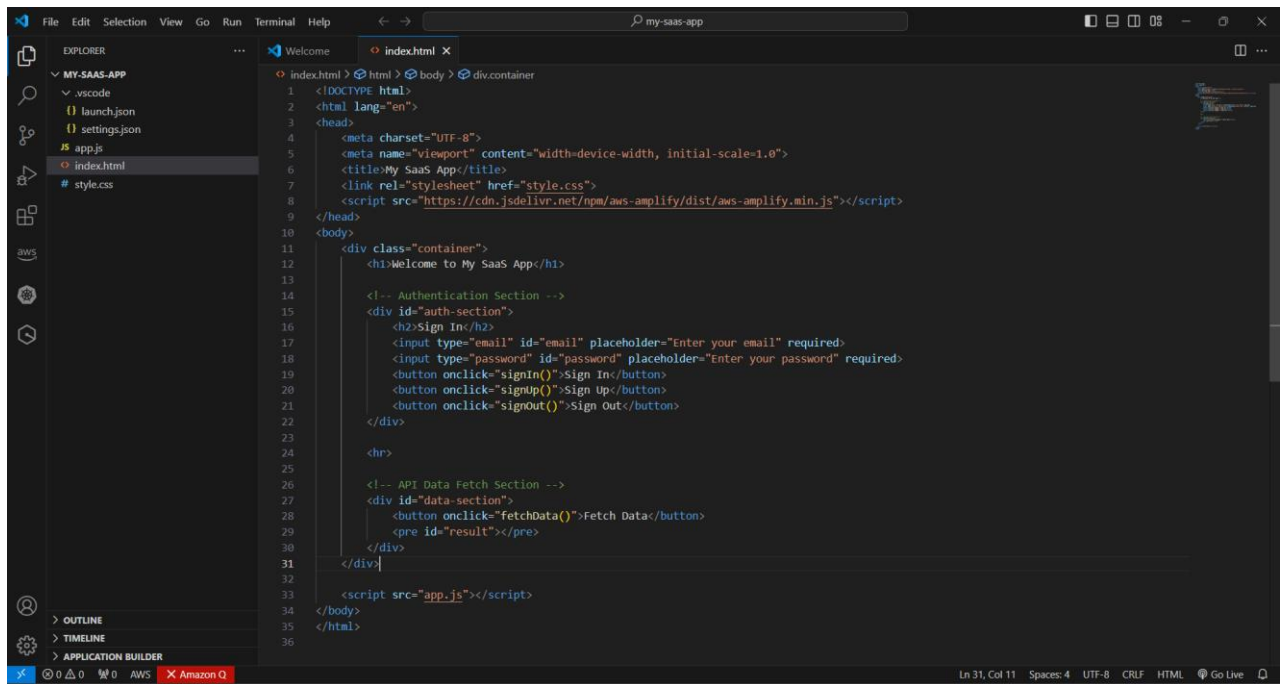
#### **3.1 HTML & CSS**

In the "Build a RESTful API for an Inventory Management System" project, HTML and CSS work in tandem to create a structured, user-friendly, and visually appealing interface that enhances the user experience while managing inventory data. HTML (Hyper Text Markup Language) forms the foundation of the user interface by defining the structure and layout of each web page. HTML elements like forms, tables, buttons, and input fields provide essential building blocks that allow users to interact with the system



**Fig 3.1.1 HTML**

CSS (Cascading Style Sheets), on the other hand, enhances the presentation and layout of these HTML elements, making the application aesthetically pleasing and intuitive to use. CSS provides styling and formatting rules that bring consistency and visual hierarchy to the interface. With CSS, elements like tables, buttons, and forms can be customized with colors, font styles, and spacing, ensuring they are visually distinct and easy to identify.



The image shows a screenshot of a Visual Studio Code editor window. The Explorer sidebar on the left shows a project named 'MY-SaaS-APP' with files like 'launch.json', 'settings.json', 'app.js', 'index.html', and 'style.css'. The main editor area displays the 'index.html' file. The code is an HTML document with a head section containing meta tags for charset and viewport, a title 'My SaaS App', and a link to 'style.css'. The body section contains a 'div.container' with a 'Welcome to My SaaS App' message, an 'Authentication Section' with sign-in and sign-up buttons, and an 'API Data Fetch Section' with a button to fetch data and a pre element for the result. The status bar at the bottom indicates 'Ln 31, Col 11', 'Spaces: 4', 'UTF-8', 'CRLF', 'HTML', and 'Go Live'.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>My SaaS App</title>
7   <link rel="stylesheet" href="style.css">
8   <script src="https://cdn.jsdelivr.net/npm/aws-amplify/dist/aws-amplify.min.js"></script>
9 </head>
10 <body>
11   <div class="container">
12     <h1>Welcome to My SaaS App</h1>
13
14     <!-- Authentication Section -->
15     <div id="auth-section">
16       <h2>Sign In</h2>
17       <input type="email" id="email" placeholder="Enter your email" required>
18       <input type="password" id="password" placeholder="Enter your password" required>
19       <button onclick="signIn()">Sign In</button>
20       <button onclick="signUp()">Sign Up</button>
21       <button onclick="signOut()">Sign Out</button>
22     </div>
23
24     <hr>
25
26     <!-- API Data Fetch Section -->
27     <div id="data-section">
28       <button onclick="fetchData()">Fetch Data</button>
29       <pre id="result"></pre>
30     </div>
31   </div>
32
33   <script src="app.js"></script>
34 </body>
35 </html>
36
```

Fig 3.1.2 HTML Code Execution.



Fig 3.1.3 CSS

CSS enables responsive design, allowing the layout to adapt to different screen sizes, making the application accessible on desktops, tablets, and smartphones. Additionally, CSS frameworks like Bootstrap can be incorporated to quickly implement standardized styling, making the design process faster and ensuring a professional look. Together, HTML and CSS create a cohesive and interactive experience for users.

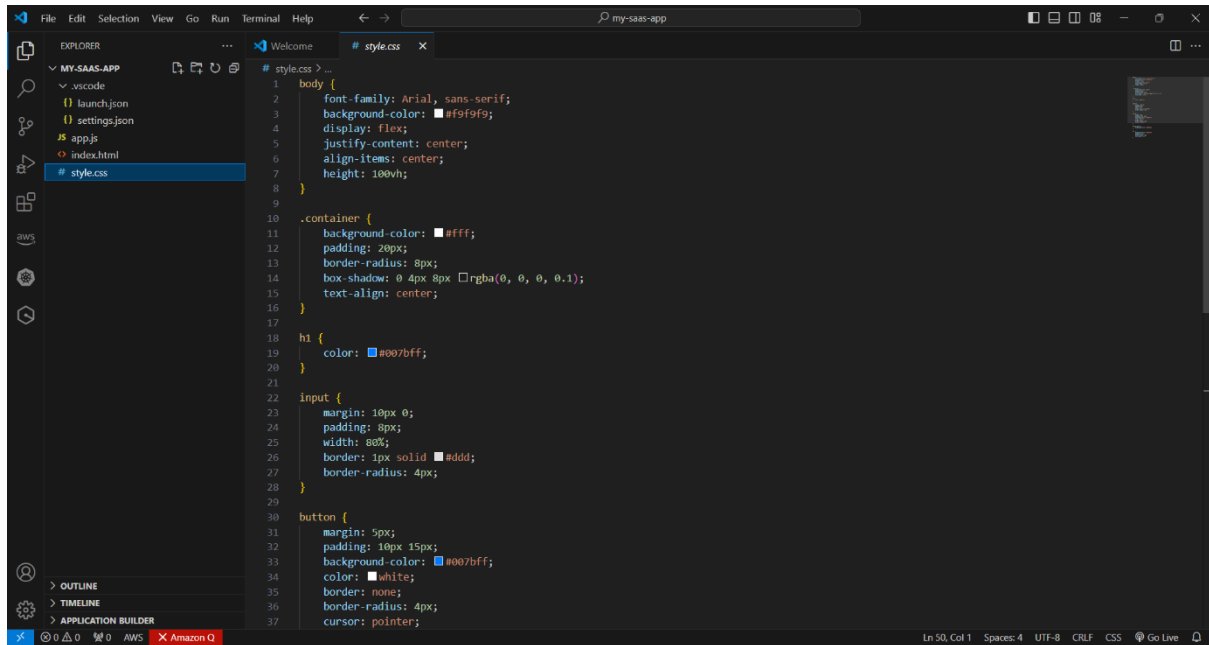
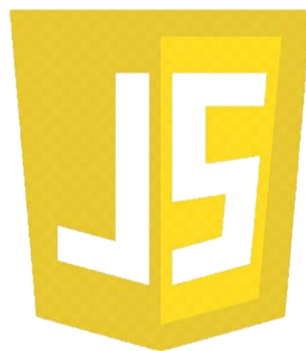


Fig 3.1.4 CSS Code Execution

## 3.2 JAVASCRIPT

Positioned as the primary scripting language on the frontend, JavaScript is responsible for managing the communication between the user interface (built with HTML and styled by CSS) and the backend RESTful API, allowing users to interact with the system seamlessly. JavaScript's primary function in this project is to handle AJAX (Asynchronous JavaScript and XML) requests, enabling asynchronous communication with the backend API.

This means that when a user performs an action, such as adding a new item, updating inventory details, or deleting an item, JavaScript can send an HTTP request to the backend without requiring a full page reload, keeping the interface responsive and user-friendly. This asynchronous communication is vital for providing a real-time, interactive experience. In addition to handling data fetching and display, JavaScript enables interactive elements that enhance usability. It powers actions like hover effects, button clicks, and form submissions, making the interface more responsive and intuitive. JavaScript frameworks and libraries like jQuery can be employed to simplify these tasks, particularly when dealing with complex animations or manipulating multiple elements at once.



**Fig 3.2.1 JavaScript**

JavaScript's flexibility also supports error handling in API calls, allowing the application to catch errors and provide feedback to the user if something goes wrong, such as network issues or invalid requests. This is particularly useful for displaying meaningful messages, like "Item added successfully" or "Error: Could not delete item." Additionally, JavaScript's interaction with CSS allows for dynamic styling adjustments, like highlighting table rows when selected or changing button colors based on user actions, making the interface feel responsive and alive. By integrating with HTML and CSS, JavaScript transforms a static web page into a fully interactive, responsive, and user-centric application, enhancing the efficiency and usability of the inventory management system.

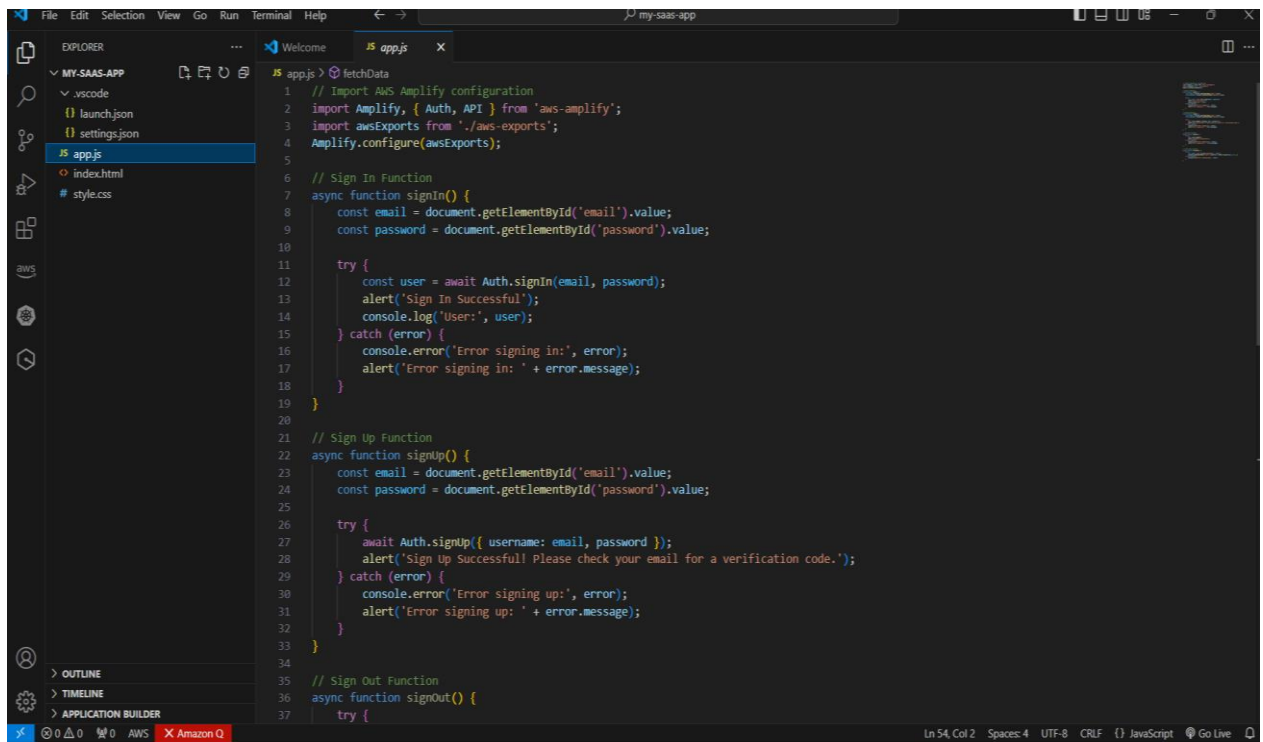


Fig 3.2.2 JS Code Execution

## CHAPTER 4

### AWS SERVICE SELECTION

When deploying a SaaS application on AWS, selecting the right services is crucial to ensure scalability, performance, security, and cost-efficiency. AWS provides a comprehensive suite of services that cater to different aspects of a cloud-based application. Below is a detailed breakdown of the AWS services selected for each layer of the SaaS application, along with their benefits and use cases. For backend services, **Amazon API Gateway** manages secure RESTful APIs, while **AWS Lambda** enables serverless computing, automatically scaling to handle varying loads without managing infrastructure. The data layer can be managed using **Amazon RDS** for relational databases or **Amazon DynamoDB** for NoSQL databases, each offering automated backups, high availability, and scaling options. **Amazon Cognito** simplifies user authentication, supporting sign-ups, sign-ins, and multi-factor authentication. Networking and security are enhanced using Amazon VPC, which isolates resources in a secure network environment. AWS CodePipeline facilitates Continuous Integration and Continuous Deployment (CI/CD), automating code builds, tests, and deployments. For monitoring and logging, Amazon CloudWatch provides real-time metrics and alerts, ensuring application health and performance. By leveraging these AWS services, businesses can build a robust, scalable, and cost-effective SaaS solution that meets modern cloud-native architecture standards.

#### 4.1 NODE.JS

Node.js plays a critical role in the API layer by serving as the backend runtime environment that powers the server-side logic. Node.js, an open-source, JavaScript-based runtime, is built on Chrome's V8 JavaScript engine and is designed for high-performance, non-blocking I/O operations, making it ideal for handling the numerous simultaneous client requests an inventory management system may receive. This efficiency is particularly valuable for our project, as Node.js enables the API to manage real-time data transactions, ensuring that users can add, retrieve, update, or delete inventory items with minimal delay. Its event-driven architecture allows the API layer to handle multiple requests concurrently, which is essential

for scalability and responsiveness, as the system needs to handle growing volumes of inventory data and user requests without sacrificing performance.

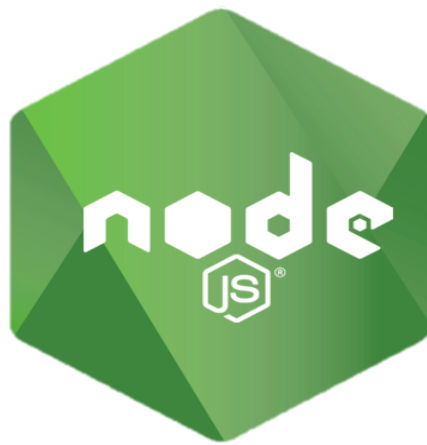
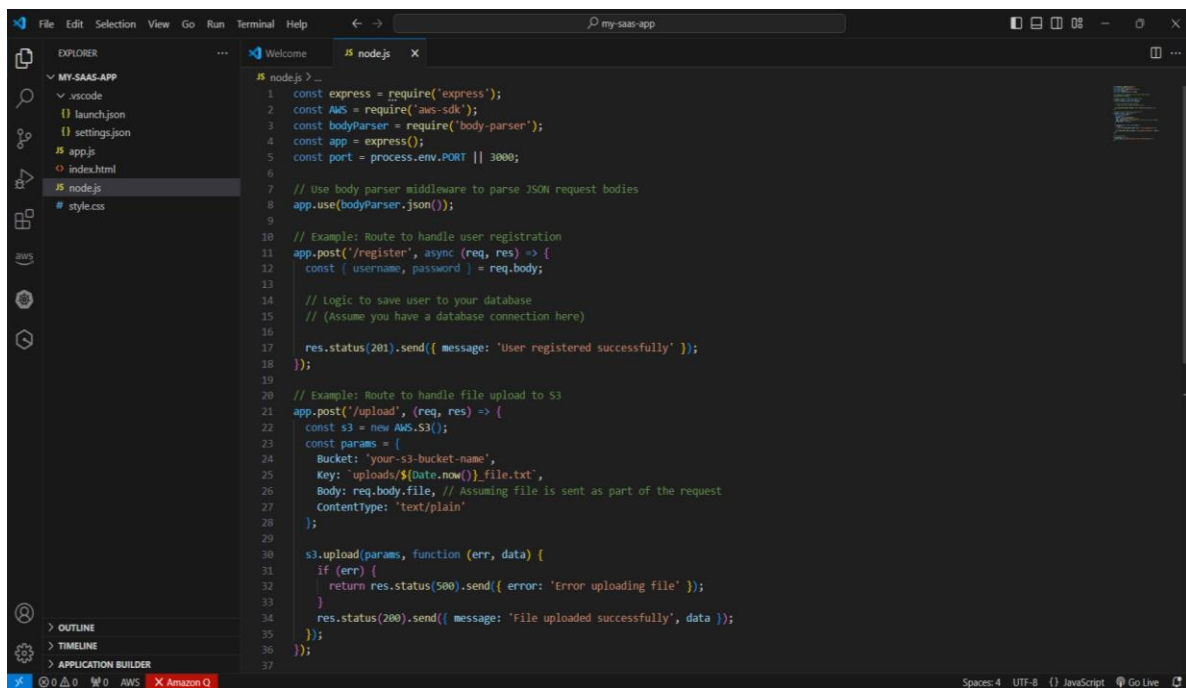


Fig 4.2.1 Node.js

A screenshot of the Visual Studio Code (VS Code) editor interface. The Explorer panel on the left shows a project named "MY-SAAS-APP" with files like "launch.json", "settings.json", "app.js", "index.html", "node.js", and "style.css". The main editor area displays the "node.js" file, which contains JavaScript code for an Express.js application. The code includes imports for 'express', 'aws-sdk', and 'body-parser', and defines routes for user registration and file upload to S3. The status bar at the bottom indicates "Spaces: 4", "UTF-8", "JavaScript", and "Go Live".

```
1 const express = require('express');
2 const AWS = require('aws-sdk');
3 const bodyParser = require('body-parser');
4 const app = express();
5 const port = process.env.PORT || 3000;
6
7 // Use body parser middleware to parse JSON request bodies
8 app.use(bodyParser.json());
9
10 // Example: Route to handle user registration
11 app.post('/register', async (req, res) => {
12   const { username, password } = req.body;
13
14   // Logic to save user to your database
15   // (Assume you have a database connection here)
16
17   res.status(201).send({ message: 'User registered successfully' });
18 });
19
20 // Example: Route to handle file upload to S3
21 app.post('/upload', (req, res) => {
22   const s3 = new AWS.S3();
23   const params = {
24     Bucket: 'your-s3-bucket-name',
25     Key: `uploads/${Date.now()}.file.txt`,
26     Body: req.body.file, // Assuming file is sent as part of the request
27     ContentType: 'text/plain'
28   };
29
30   s3.upload(params, function (err, data) {
31     if (err) {
32       return res.status(500).send({ error: 'Error uploading file' });
33     }
34     res.status(200).send({ message: 'File uploaded successfully', data });
35   });
36 });
37
```

Fig 4.2.2 node JS Code Execution

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to build server-side applications using JavaScript. Traditionally, JavaScript was confined to the browser, but Node.js enables developers to use JavaScript on the server, making it possible to build the entire stack (both frontend and backend) with a single programming language. Node.js is popular for its performance, scalability, and its vast ecosystem, making it an excellent choice for building modern web applications, including REST APIs, real-time applications, and microservices.

**1. Environment Configuration:**

Sensitive data, such as database credentials and API keys, should be managed using environment variables, often stored in .env files and loaded using packages like dotenv.

**2. Data Validation and Sanitization:**

Libraries like Joi and validator help validate and sanitize input data, ensuring security against attacks like SQL injection and XSS (Cross-Site Scripting).

**3. Authentication and Authorization:**

Node.js supports authentication using methods like JSON Web Tokens (JWT), OAuth, and sessions, ensuring only authorized users can access protected routes.



## CHAPTER 5

### INTERFACE OF THE PROJECT

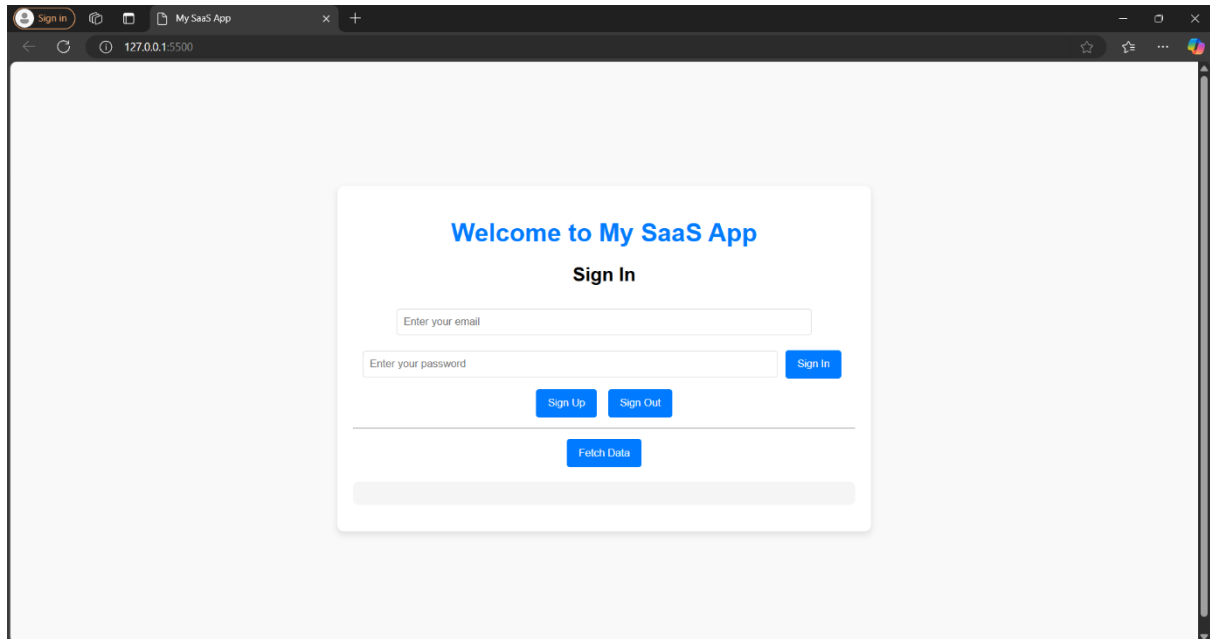


Fig 5.1 STARTING INTERFACE

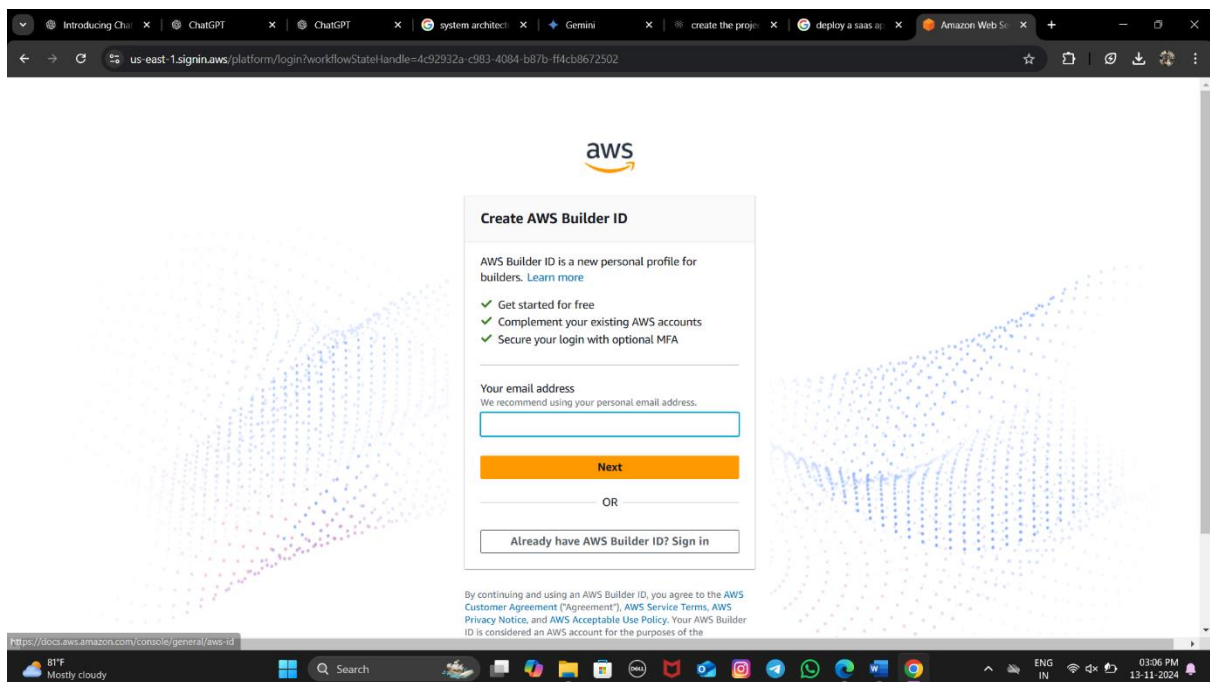


Fig 5.1 INPUT LISTED INTERFACE

```
1 output "state_infrastructure_outputs" {
2
3     //grouping output by state
4     value = {
5         for state, infrastructure in module.aws_humangov_infrastructure :
6             state => {
7             ec2_public_dns = infrastructure.state_ec2_public_dns
8             dynamodb_table = infrastructure.state_dynamodb_table
9             s3_bucket      = infrastructure.state_s3_bucket
10         }
11     }
12 }
```

Fig 5.3 AWS OUTPUT STORED INTERFACE

## **CHAPTER 6**

### **CONCLUSION**

In conclusion, Node.js offers a powerful, efficient, and versatile environment for building modern web applications and backend services. Its non-blocking, event-driven architecture, powered by the V8 JavaScript engine, makes it highly suitable for I/O-intensive applications requiring concurrent connections, such as RESTful APIs and real-time applications. Node.js supports seamless integration with SQL and NoSQL databases, allowing it to handle various data management needs. With a vast ecosystem of libraries and tools through NPM, developers have access to pre-built solutions for everything from data validation to real-time communication, accelerating development. Security and scalability are well-supported in Node.js, with options for environment configuration, role-based access control, data encryption, and clustering to optimize server utilization. The built-in module system promotes modular coding, while popular frameworks like Express simplify web server and API development. Moreover, Node.js has robust testing, debugging, and deployment support, making it reliable for both small and large-scale applications. Its compatibility with microservices, serverless architectures, and cloud platforms like AWS and Azure further enhances its flexibility and scalability. Overall, Node.js enables developers to build highly efficient, scalable, and maintainable applications that meet diverse performance and operational demands.



**BALA KUMAR**

is here by awarded the certificate of achievement  
for the successful completion of

**First SaaS Application**

A handwritten signature in blue ink, reading 'M. Arunprakash'.

**M. Arunprakash**

Founder and CEO,  
GUVI Geek Networks.

Certificate issued on : October 24 2024

Verified certificate ID: 1vj6772t19847Rq721

Verify certificate at: [www.guvi.in/certificate?id=1vj6772t19847Rq721](http://www.guvi.in/certificate?id=1vj6772t19847Rq721)



**Balaji Marimuthu**

is here by awarded the certificate of achievement  
for the successful completion of

**First Saas Application**

A handwritten signature in blue ink, reading 'M. Arunprakash'.

**M. Arunprakash**

Founder and CEO,  
GUVI Geek Networks.

Certificate issued on : **November 7 2024**

Verified certificate ID: **LS58035Z005hdN179U**

Verify certificate at: [www.guvi.in/certificate?id=LS58035Z005hdN179U](http://www.guvi.in/certificate?id=LS58035Z005hdN179U)