SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report
on
InstaNote
Submitted in Partial fulfillment of the requirements for the award of the Degree of

Bachelor of Science (Cloud Computing and Big Data)

Submitted by

Jayram Dhawal and Raj

R22DB017 and R22DB031

Under the guidance of

Dr.Devi A
Internal Guide

Jan 2025

Rukmini Knowledge Park, Kattigenahalli, Yelahanka, Bengaluru-560064
www.reva.edu.in

REVA
UNIVERSITY
Bengaluru, India

SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

CERTIFICATE

The project work titled **InstaNote,** is being carried out under my guidance by Jayram Dhawal and Raj, SRN: R22DB017 and R22DB031, a Bonafide students at REVA University, and are submitting the project report in partial fulfillment, for the award of **Bachelor of Science (CC & BD) in Computer Science –Cloud Computing and Big Data** during the academic year **2024–25**. The project report has been approved, as it satisfies the academic requirements with respect to the project work prescribed for the forementioned Degree.

[illegible]
6/1/25
Signature with date

Dr.Devi A
Internal Guide

[illegible] 6/1/25
Signature with date

Dr.G.Sasikala
Head of the Department-BSc

[illegible] 6.1.24
Signature with Date

Dr C K Lokesh
Director

Name of the Examiner with Affiliation	**Signature with Date**
1. [illegible]	08/01/25
2. [illegible] 8/1/25	

DECLARATION

We Mr. Jayram Dhawal and Raj, pursuing our **Bachelor of Science (Cloud Computing and Big Data)**, offered by School of Computer Science and Applications, Reva University, declare that this Project titled "InstaNote", is the result of the project work done by us under the supervision of Dr.Devi A.

We are submitting this Project Work in partial fulfillment of the requirements for the award of the degree of **Bachelor of Science (CC & BD)** by REVA University, Bengaluru, during the Academic Year 2024-25.

We further declare that this Project Report or any part of it has not been submitted for the award of any other Degree / Diploma of this University or any other University/ Institution.

[illegible] [illegible]
(Signature of the candidate)

Signed on: 07-01-2025

Certified that this project work submitted by Jayram Dhawal and Raj has been carried out under my guidance and the declaration made by the candidate is true to the best of my knowledge.

[illegible] 6/1/24
Signature of Internal Guide
Date :..6/1/24

[illegible]
Signature of External
Date :.06/01/25

[illegible] C.K.

Signature of Director of the School
Date :………..
Official Seal of the School

DIRECTOR
School of Computer Science & Applications
REVA University, Kattigenahalli,
Bangalore - 560 064.

ACKNOWLEDGEMENT

We hereby acknowledge all those, under whose support and encouragement, we have been able to fulfil all our academic commitments successfully. In this regard, we take this opportunity to express our deep sense of gratitude and sincere thanks to School of Computer Science and Applications which has always been a tremendous source of guidance.

We express our sincere gratitude to Dr. P. SHYAMA RAJU, Honorable Chancellor, REVA University, and Bengaluru for providing us the state-of-the-art facilities.

We are thankful to **Dr. SANJAY CHITNIS**, Vice Chancellor, REVA University, **Dr K S NARAYANASWAMY,** Registrar, REVA University, Bengaluru for their support and encouragement.

We take this opportunity to express our heartfelt thanks to **Dr C K LOKESH**, Director, School of CSA, REVA University and our sincere thanks to **Dr. G. SASIKALA** Associate Professor, Head of the Department for BSc Program, School of CSA, REVA University, whose encouragement and best wishes provided impetus for the Project Work carried out.

Also, our sincere gratitude goes to our internal guide, Dr.Devi A, Associate Professor, for the valuable suggestion and constant encouragement towards the completion of this project.

Last, but not the least, We thank our parents for their incredible support and encouragement throughout.

ABSTRACT

In the digital age, secure and efficient file sharing is paramount. This mini project focuses on developing a robust InstaNote leveraging Amazon Web Services (AWS) and its ecosystem. The core technologies employed in this project include AWS, Amazon S3, HTML, CSS, JavaScript, and Node.js.

The project aims to provide users with a seamless experience for uploading, storing, and sharing files over the internet. By utilizing AWS, the platform ensures scalability, reliability, and security. Amazon S3 (Simple Storage Service) is employed as the primary storage solution due to its durability and ease of integration.

The front-end of the application is built using HTML, CSS, and JavaScript, ensuring a responsive and user-friendly interface. HTML forms the structure, CSS enhances the visual appeal, and JavaScript adds interactivity to the platform. These technologies work in tandem to provide a smooth and intuitive user experience.

On the server-side, Node.js is used to handle the back-end logic. Node.js, known for its efficiency and performance, manages the interaction between the front-end and AWS services. It handles file uploads, communicates with Amazon S3 for storing and retrieving files, and ensures secure access control.

The platform features user authentication to ensure that only authorized users can upload or access files. This is achieved through a combination of session management and AWS Identity and Access Management (IAM) policies. Security measures, including encryption and secure data transfer protocols, are implemented to protect user data.

This project demonstrates the power of AWS in building scalable and secure web applications. It highlights the integration of various web technologies to create a functional and efficient InstaNote. The use of Amazon S3 as a storage solution showcases its capabilities in handling large volumes of data with high availability.

Overall, this mini project serves as a practical example of how modern web technologies can be leveraged to address real-world problems in file sharing and storage. The platform is designed with user experience and security at its core, providing a reliable solution for individuals and organizations alike.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Introduction to Project

In today's digitally interconnected world, the ability to share files efficiently and securely is essential for both individuals and organizations. This mini project focuses on the development of a versatile InstaNote using a suite of modern web technologies, with Amazon Web Services (AWS) serving as the backbone. The primary objective is to create a scalable, reliable, and user-friendly platform that allows users to upload, store, and share files seamlessly.

The project leverages AWS, renowned for its robust cloud computing capabilities, to ensure the platform's scalability and security. Amazon S3 (Simple Storage Service) is utilized as the main storage solution due to its high availability, durability, and cost-effectiveness. This choice enables the platform to handle large volumes of data while ensuring data integrity and accessibility.

The front-end of the platform is constructed using HTML, CSS, and JavaScript. HTML provides the structural framework for the web pages, while CSS is employed to enhance the visual aesthetics, ensuring a responsive and visually appealing user interface. JavaScript adds interactivity to the platform, allowing for dynamic content updates and a smoother user experience.

Node.js, a powerful and efficient server-side technology, forms the core of the back end. It facilitates the server-side scripting, enabling the platform to handle multiple simultaneous connections with minimal overhead. Node.js is responsible for managing file uploads, communicating with Amazon S3 for storing and retrieving files, and ensuring secure access to the stored data.

1.2 Statement of the Problem

Traditional file sharing methods often suffer from limitations such as security vulnerabilities, scalability issues, and inefficient access controls. Users face challenges in securely storing and sharing large volumes of files, especially when dealing with sensitive information. The existing solutions may lack advanced features and robust security mechanisms, leading to potential data breaches and unauthorized access. This project addresses these issues by developing a cloud-based InstaNote that ensures high availability, secure data transmission and storage, and efficient access management.

1.3 Brief Description of the Project

The InstaNote project aims to create a web-based application that allows users to upload, store, and share files securely and efficiently. The platform incorporates user authentication, permissions management via AWS IAM, and encryption to safeguard data. Additionally, it provides features for sharing files with specific users and tracking user activity.

1.4 Software and Hardware Specification

- **Software:**
    - **AWS (Amazon Web Services):** Cloud infrastructure and storage services.
    - **Amazon S3 (Simple Storage Service):** Secure and scalable file storage.
    - **HTML:** Markup language for structuring web content.
    - **CSS:** Styling language for designing the user interface.
    - **JavaScript:** Scripting language for adding interactivity and dynamic content.
    - **Node.js:** Server-side runtime environment for backend development.
- **Hardware:**
    - **Client Devices:** Any device with a web browser, such as desktops, laptops, tablets, or smartphones.
    - **Server:** AWS cloud servers, providing scalability and high availability for hosting the platform.

1.5 Functional and Non-Functional Requirements

Functional Requirements:

1. **User Registration and Authentication:**
    - Users must be able to create an account on the platform by providing necessary details such as username, email, and password.
    - Secure authentication mechanisms must be in place to ensure that users can log in and access their accounts securely. This includes using techniques such as hashing passwords and implementing multi-factor authentication (MFA).

2. **File Upload and Retrieval:**
    - Users should be able to upload files to the platform. This functionality includes selecting files from their local device and uploading them to the server.
    - Once uploaded, users must be able to retrieve and download their files. The platform should provide an interface for viewing and downloading stored files.

3. **Secure File Storage in AWS S3:**
    - Files uploaded by users must be securely stored in Amazon S3. AWS S3 provides scalable storage with high durability and availability.
    - The platform should implement encryption for files both in transit and at rest to ensure data security.

Non-Functional Requirements:

1. **High Availability and Scalability:**
    - The platform must be designed to handle many users and file uploads without downtime. Using AWS infrastructure, the platform can leverage services like AWS Elastic Load Balancing and Auto Scaling to ensure high availability and scalability.

2. **Secure Data Transmission and Storage:**
    - Data must be transmitted securely between the user's device and the server using HTTPS. SSL/TLS certificates should be used to encrypt data in transit.
    - Data stored in S3 should be encrypted using AWS-managed keys or customer-managed keys for added security.

3. **Intuitive User Interface:**
    - The platform should have a user-friendly interface that is easy to navigate. The design should be responsive, ensuring a seamless experience across different devices and screen sizes.
    - Clear instructions and feedback messages should guide users through the file upload and retrieval process.

4. **Fast Response Time for File Uploads and Downloads:**
    - The platform must provide quick and efficient file uploads and downloads. This can be achieved by optimizing server-side processing and using Content Delivery Networks (CDNs) to cache and deliver files closer to the user.
    - Performance monitoring and optimization techniques should be implemented to minimize latency and ensure a smooth user experience.

5. **Backup and Disaster Recovery:**
    - Implement backup mechanisms to ensure data can be restored in case of accidental deletion or disasters.
    - Utilize AWS services like S3 Versioning and AWS Backup to maintain data integrity and recovery options.
6. **Compliance with Data Privacy Regulations:**
    - Ensure the platform complies with relevant data privacy regulations, such as GDPR or CCPA.
    - Implement measures to protect user data and provide transparency regarding data handling practices.

1.6 Company Profile

InstaNote was conceptualized to bridge the gap between convenience and security in the file-sharing domain. The team focuses on leveraging modern web technologies and cloud services to provide a reliable solution for users worldwide.

2.Literature Survey:

1. **Cloud Storage and Its Advantages:**
   - Various studies have highlighted the advantages of cloud storage services like Amazon S3. Cloud storage offers scalability, flexibility, and cost efficiency compared to traditional storage solutions. Articles and research papers discuss how businesses and developers leverage cloud services to manage large volumes of data without worrying about hardware limitations or maintenance. The use of Amazon S3, in particular, has been praised for its durability and availability, making it an ideal choice for InstaNotes.
2. **Web Development Technologies:**
   - The evolution of web development technologies has been well-documented in literature. HTML, CSS, and JavaScript form the cornerstone of front-end development, offering a structure, styling, and interactivity for web applications. Research papers and books discuss best practices in web design and development, emphasizing the importance of creating responsive and user-friendly interfaces. The role of JavaScript in adding dynamic content and enhancing user experience is also a common topic of discussion.
3. **Security in Cloud-Based Applications:**
   - Security is a critical aspect of any web application, especially those involving file storage and sharing. Literature on cloud security covers topics such as encryption, secure access controls, and data protection best practices. Articles and white papers provide insights into implementing security measures, such as using IAM policies for access management in AWS and ensuring data is encrypted both in transit and at rest. Case studies often illustrate the consequences of inadequate security measures and the importance of proactive security planning.
4. **Scalability and Performance Optimization:**
   - The ability to scale applications to handle varying loads is essential for cloud-based platforms. Literature on scalability and performance optimization explores techniques such as auto-scaling, load balancing, and caching. Research papers discuss how services like AWS Elastic Load Balancing and Amazon CloudFront can be used to distribute traffic and reduce latency. Performance monitoring tools and strategies for optimizing file uploads and downloads are also common topics in this area.

3. System Analysis

3.1 Existing System

1. **Traditional File Storage Solutions:**
    - Traditionally, file storage and sharing were managed using on-premises servers and storage devices. These systems required substantial investment in hardware, maintenance, and security, making them costly and complex to manage. The scalability was limited by the physical infrastructure, often leading to performance bottlenecks and storage constraints.

2. **Limited Accessibility and Collaboration:**
    - Existing file sharing solutions often lack robust access and collaboration features. Users had to rely on manual methods like email attachments or physical transfers to share files. This process was inefficient, time-consuming, and prone to data loss or corruption. Additionally, it was challenging to provide secure and controlled access to multiple users, impacting collaboration and productivity.

3. **Security Concerns:**
    - Traditional file storage solutions faced significant security challenges. Protecting data from unauthorized access, breaches, and malware required continuous monitoring and updates. Many existing systems lacked advanced security measures such as encryption, making sensitive data vulnerable to attacks. This increased the risk of data breaches and compliance issues.

4. **Lack of Scalability:**
    - Scalability was a significant limitation in traditional systems. As the volume of data grew, organizations struggled to scale their storage infrastructure effectively. Adding new hardware was not only expensive but also time-consuming. This limited the ability to handle large volumes of data and adapt to changing business needs quickly.

5. **Inefficient Resource Utilization:**
    - On-premises solutions often resulted in inefficient resource utilization. Servers and storage devices had fixed capacities, leading to either underutilization or over-provisioning. Maintaining and managing these resources required dedicated IT staff and constant monitoring, diverting resources from core business activities.

3.2 Limitations of the Existing System

1. **High Cost of Maintenance:**
    - Traditional file sharing systems often rely on on-premises hardware and infrastructure, leading to significant maintenance costs. Organizations need to invest in physical storage devices, servers, and network equipment, which require regular updates and repairs. This can be a financial burden, especially for small to medium-sized enterprises.

2. **Scalability Issues:**
    - Existing systems may struggle with scalability, particularly when dealing with an increasing volume of data and users. On-premises solutions have fixed storage capacities, and expanding these resources can be challenging and expensive. As a result, organizations may face performance bottlenecks and limited storage availability as their needs grow.

3. **Limited Accessibility and Collaboration:**
    - Traditional file sharing methods often lack robust features for remote access and collaboration. Users may have difficulty accessing files from different locations or devices, hindering productivity and collaboration. Additionally, manual file sharing processes, such as email attachments, can be inefficient and prone to errors.

    ○

4. **Security Vulnerabilities:**
    - Ensuring the security of sensitive data is a major concern for existing systems. Traditional file sharing solutions may lack advanced security measures, such as encryption and access control, making them vulnerable to data breaches, unauthorized access, and malware attacks. Protecting data at rest and in transit is crucial for maintaining confidentiality and integrity.

5. **Resource Inefficiency:**
    - On-premises file storage systems often lead to inefficient resource utilization. Storage devices and servers may be underutilized or over-provisioned, resulting in wasted resources and increased operational costs. Organizations must allocate dedicated IT staff to manage and monitor these resources, diverting attention from core business activities.

3.3 Proposed System

1. **Cloud-Based Solution with AWS:**
    - The proposed system leverages the power of Amazon Web Services (AWS) to create a cloud-based InstaNote. This cloud infrastructure ensures scalability, high availability, and disaster recovery. By using AWS, the platform can easily scale to meet the demands of increasing users and file storage requirements without the need for physical hardware investments.

2. **Secure and Durable Storage with Amazon S3:**
    - Amazon S3 (Simple Storage Service) is employed as the main storage solution, providing highly durable and secure storage for user files. S3's features, such as data encryption and access control, ensure that files are stored securely. Additionally, S3's high availability guarantees that users can access their files anytime, from anywhere.

3. **User-Friendly Front-End Development:**
    - The front-end of the platform is built using HTML, CSS, and JavaScript, ensuring a responsive and intuitive user interface. HTML structures the content, CSS enhances the visual appeal, and JavaScript adds interactivity. This combination ensures a seamless user experience across various devices, including desktops, tablets, and smartphones.

4. **Efficient Back-end with Node.js:**
    - Node.js is utilized for back-end development, handling server-side logic and managing interactions with AWS services. Node.js' non-blocking, event-driven architecture ensures efficient handling of multiple concurrent requests, providing a smooth and responsive user experience. It manages file uploads, user authentication, and secure data transactions.

5. **Enhanced Security and Access Management:**
    - The proposed system incorporates robust security measures to protect user data. User authentication is implemented to ensure only authorized access to the platform. Permissions management is handled using AWS Identity and Access Management (IAM), allowing for fine-grained control over who can access and manage files. Encryption is applied to data both in transit and at rest to maintain confidentiality and integrity.

3.4 Advantages of the Proposed System

1. **Scalability and Flexibility:**
    - The proposed system leverages AWS, providing the ability to scale seamlessly with user demand. Whether accommodating a growing user base or increasing data volumes, AWS services like Amazon S3 and Elastic Load Balancing ensure that the platform can scale efficiently without compromising performance. This flexibility allows the platform to adapt to changing needs and handle large-scale operations with ease.

2. **Enhanced Security:**
    - Security is a top priority in the proposed system. By utilizing AWS Identity and Access Management (IAM) for permissions management and implementing encryption for data both in transit and at rest, the platform ensures that user data is protected from unauthorized access and breaches. This robust security framework enhances user trust and complies with industry standards and regulations.

3. **Cost Efficiency:**
    - Using AWS eliminates the need for significant upfront investments in physical hardware and reduces ongoing maintenance costs. The pay-as-you-go model of AWS allows for cost-effective scaling, where users only pay for the resources, they actually use. This makes the platform financially accessible for both small and large organizations, providing a cost-efficient solution for file sharing and storage.

4. **User-Friendly Interface:**
    - The proposed system offers a responsive and intuitive user interface designed with HTML, CSS, and JavaScript. This ensures a positive user experience, making it easy for users to upload, manage, and share files. The use of modern web technologies enhances the platform's accessibility across different devices and screen sizes, catering to a wide range of users.

5. **High Availability and Reliability:**
    - By leveraging AWS's robust infrastructure, the proposed system ensures high availability and reliability. Services like Amazon S3 provide durable storage with minimal downtime, while AWS's global network of data centers ensures that files are accessible from anywhere in the world. This high level of reliability ensures that users can depend on the platform for their file sharing needs without interruptions.

3.5 Feasibility Study

Technical Feasibility

1. **Availability of Technology:**
   - The technologies required for the project, such as AWS, Amazon S3, HTML, CSS, JavaScript, and Node.js, are readily available and well-documented. These technologies are widely used and supported, ensuring a strong foundation for development.

2. **Skill Set of Development Team:**
   - The development team needs to have proficiency in the specified technologies. Familiarity with cloud computing, web development, and server-side programming is essential. Training and resources are available to help the team gain any additional skills required.

3. **Integration Capabilities:**
   - The proposed system integrates well with existing AWS services, ensuring seamless communication between components. Amazon S3's APIs and SDKs make it easy to integrate with Node.js for file storage and retrieval, while front-end technologies like HTML, CSS, and JavaScript can be effectively used to build a responsive interface.

4. **Scalability and Performance:**
   - AWS offers scalable infrastructure that can handle varying loads efficiently. Services like AWS Elastic Load Balancing and Auto Scaling ensure that the platform can scale as needed, maintaining performance even with a growing user base and data volume.

5. **Security Measures:**
   - The technical feasibility of implementing robust security measures is high. AWS provides various security features, such as IAM for access control, encryption for data protection, and compliance with industry standards. These measures can be effectively integrated into the platform to ensure data security.
6. **Ease of Integration:**
   - Node.js, being a server-side JavaScript runtime, allows for easy integration with front-end technologies (HTML, CSS, JavaScript). AWS SDKs simplify interactions with Amazon S3, making it straightforward to implement file storage and retrieval functionalities.

Economical Feasibility

1. **Cost of Development:**
   - The initial development costs include expenses related to AWS services, development tools, and potentially training for the development team. However, the overall cost is lower compared to setting up and maintaining on-premises infrastructure.

2. **Operational Costs:**
   - The pay-as-you-go model of AWS allows for cost-effective operations. Organizations only pay for the resources they use, reducing unnecessary expenditure. Additionally, AWS's pricing model provides flexibility, making it economical for both small and large-scale operations.

3. **Return on Investment (ROI):**
   - The platform's ability to provide secure and efficient file sharing can attract a wide user base, potentially generating revenue through subscription models or advertisements. The scalability and reliability offered by AWS ensure that the investment is justified with long-term benefits.

4. **Maintenance Costs:**
   - AWS handles much of the infrastructure maintenance, reducing the need for a large in-house IT team. This lowers ongoing maintenance costs and ensures that the platform remains up to date with minimal effort.

5. **Cost Savings:**
   - By leveraging cloud services, the project avoids significant upfront investments in hardware and reduces operational overhead. This results in substantial cost savings over traditional on-premises solutions.

6. **Resource Allocation:**
   - The project requires minimal hardware investment since it leverages cloud services. Development costs primarily involve time and effort in learning and implementing the technologies, which can be balanced with other commitments.

7. **Return on Investment (ROI):**
   - By investing in this project, you gain valuable experience and skills in cloud computing, web development, and server-side programming. This can enhance your professional profile and open up new career opportunities, providing long-term ROI.

Operational Feasibility

1. **Ease of Use:**
   - The user-friendly interface developed with HTML, CSS, and JavaScript ensures that users can easily navigate the platform, upload, manage, and share files. Clear instructions and feedback mechanisms enhance the user experience.

2. **User Training and Support:**
   - Minimal training is required for users to become familiar with the platform. Support resources, such as tutorials and help documentation, can be provided to assist users in understanding and utilizing the platform effectively.

3. **Operational Workflow:**
   - The platform streamlines the file sharing process, reducing manual effort and improving efficiency. Automated processes, such as file uploads and permissions management, simplify operations and enhance productivity.

4. **Reliability and Availability:**
   - AWS's robust infrastructure ensures high availability and reliability of the platform. Services like Amazon S3 and AWS Elastic Load Balancing minimize downtime, ensuring that users can access their files whenever needed.

5. **Support and Maintenance:**
   - AWS provides comprehensive support and maintenance for its services, ensuring that any technical issues are resolved promptly. Regular updates and security patches are applied, keeping the platform secure and up to date.

6. **User Training and Documentation:**
   - Providing clear documentation and user training materials will ensure that users can quickly learn how to use the platform effectively. This will reduce the learning curve and increase user satisfaction.

7. **Flexibility:**
   - The platform can be easily modified and extended to include additional features or support different use cases. This flexibility allows for continuous improvement and adaptation to changing user needs.

4. SYSTEM DESIGN AND DEVELOPMENT

4.1 HIGH LEVEL DESIGN (ARCHITECTURAL)

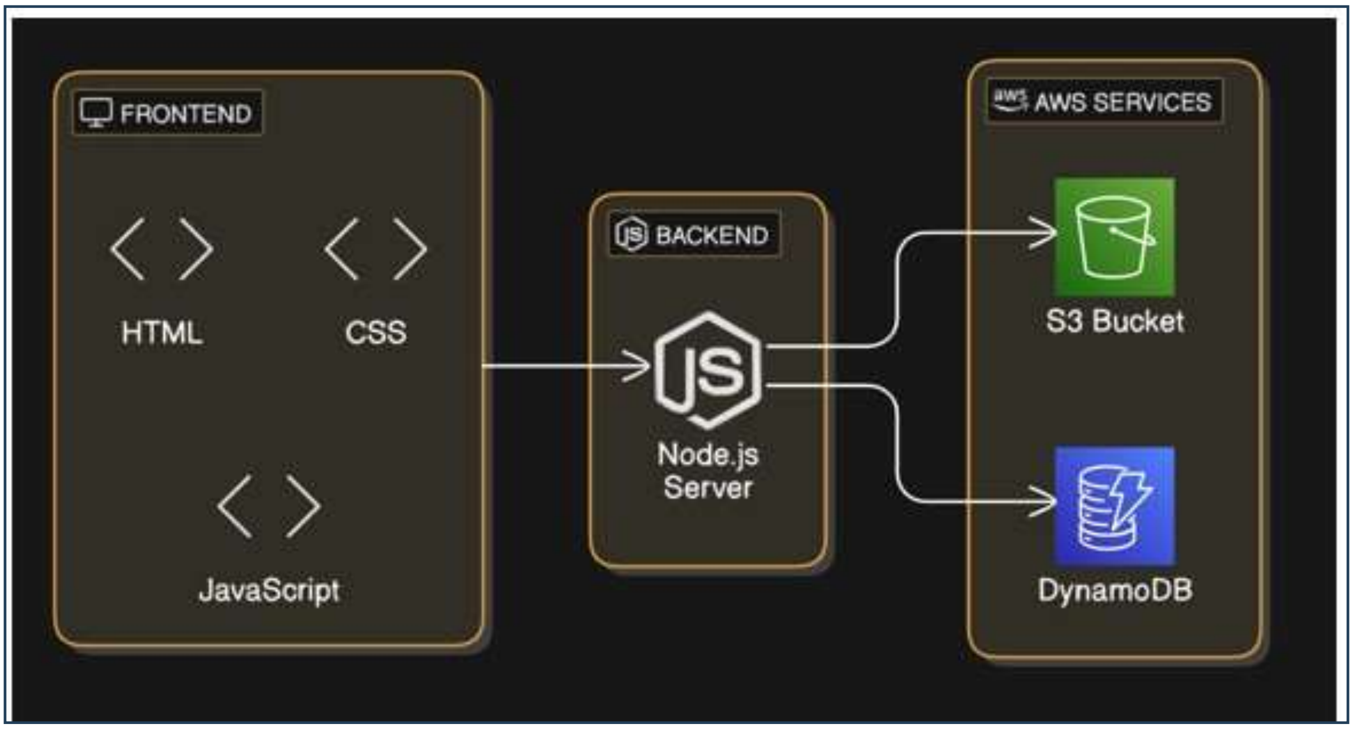


Fig.1: High Level Diagram

Architecture Overview:
The image illustrates a web application architecture with three main components: Frontend, Backend, and AWS Services.

Frontend Stack:
The frontend consists of HTML, CSS, and JavaScript, responsible for creating the user interface and handling interactions.

Backend Stack:
The backend is powered by a Node.js server, which acts as the intermediary between the frontend and the backend services.

AWS Services Integration:
The architecture utilizes AWS S3 Bucket for storing files or static assets and AWS DynamoDB for managing the application's database requirements.

Data Flow:
The frontend communicates with the backend via API calls, and the backend interacts with AWS services (S3 Bucket and DynamoDB) to retrieve or store data as needed.

4.2 LOW LEVEL DESIGN

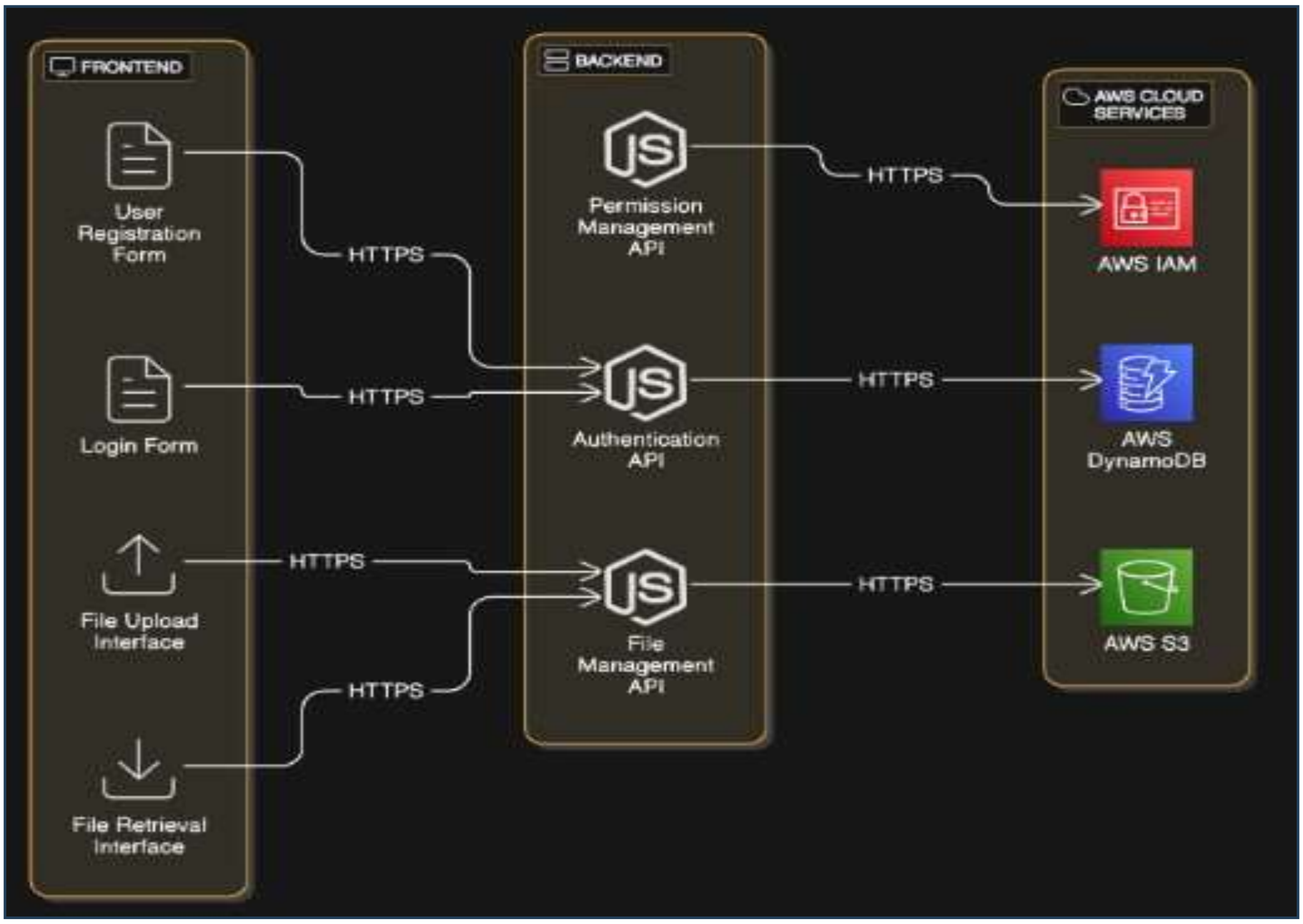


Fig.2: Low Level Diagram

Modular Architecture:
The system is divided into three components: **Frontend**, **Backend**, and **AWS Cloud Services**. The front end provides forms for user interaction (registration, login, file upload, and file retrieval), while the backend handles the APIs for authentication, permissions, and file management, with cloud services facilitating storage and access control.

Backend API Functions:

- Authentication API: Handles user login and authentication by communicating with AWS DynamoDB.
- Permission Management API: Manages user permissions by interacting with AWS IAM.
- File Management API: Manages file uploads and retrievals, storing and retrieving data from AWS S3.
-

Secure Communication:
All interactions between the frontend, backend, and AWS services occur over **HTTPS**, ensuring secure data transmission throughout the system.

4.3 DATAFLOW DIAGRAM

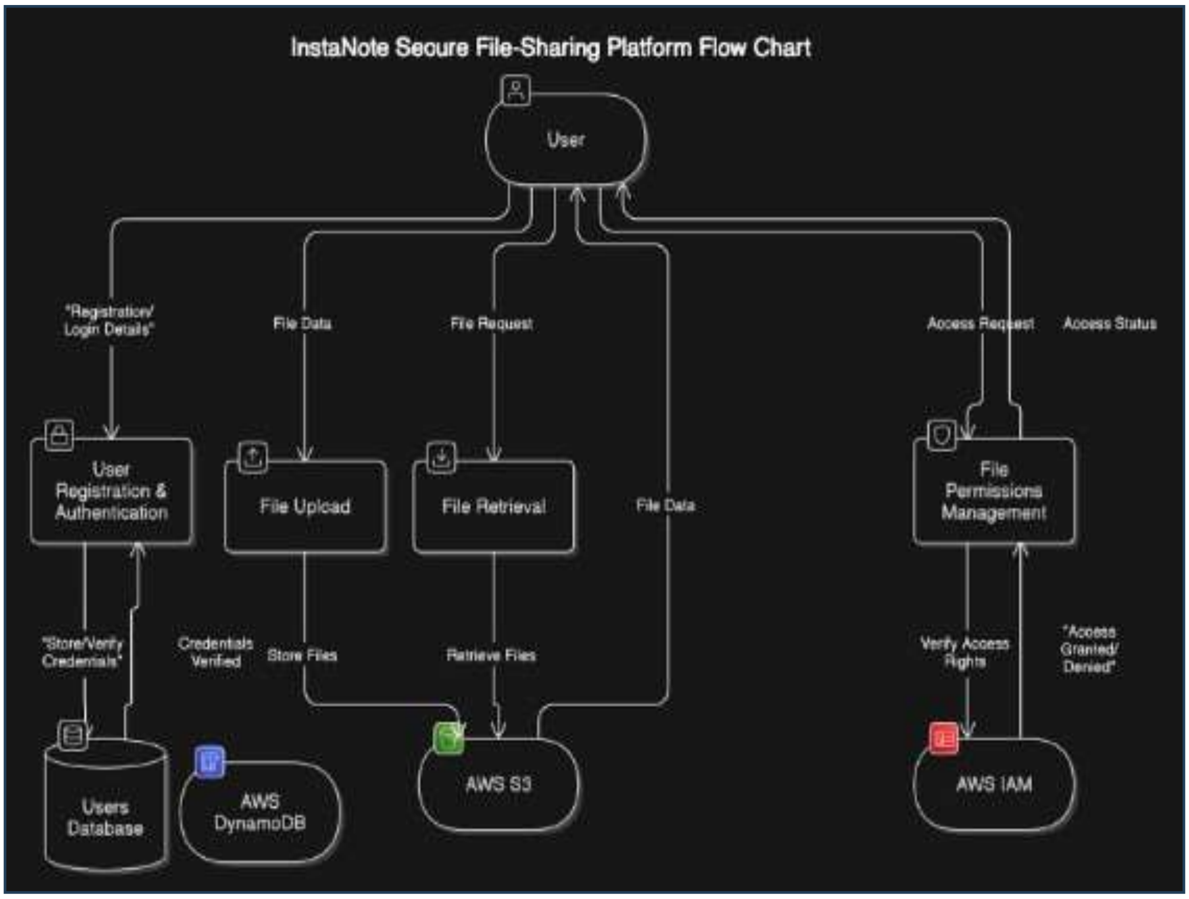


Fig.3: Dataflow Diagram

Platform Overview:
The flow chart represents the InstaNote Secure File-Sharing Platform, designed to enable user registration, authentication, file upload, file retrieval, and permissions management in a secure manner.

Core Functionalities:
- User Registration & Authentication: Users register or log in, with their credentials stored and verified using AWS DynamoDB.
- File Upload: Users upload files, which are stored in AWS S3.
- File Retrieval: Users can retrieve files from AWS S3 upon request.
- Permissions Management: Access rights are verified using AWS IAM to determine whether the user can access specific files.

Data Flow:
- User details are passed to the User Registration & Authentication module, which verifies and stores credentials in AWS DynamoDB.
- Uploaded files are sent to AWS S3, and requests for file retrieval are processed from the same storage.
- Permissions management ensures users have proper access rights, consulting AWS IAM for verification.

4.4 USE CASE DIAGRAM

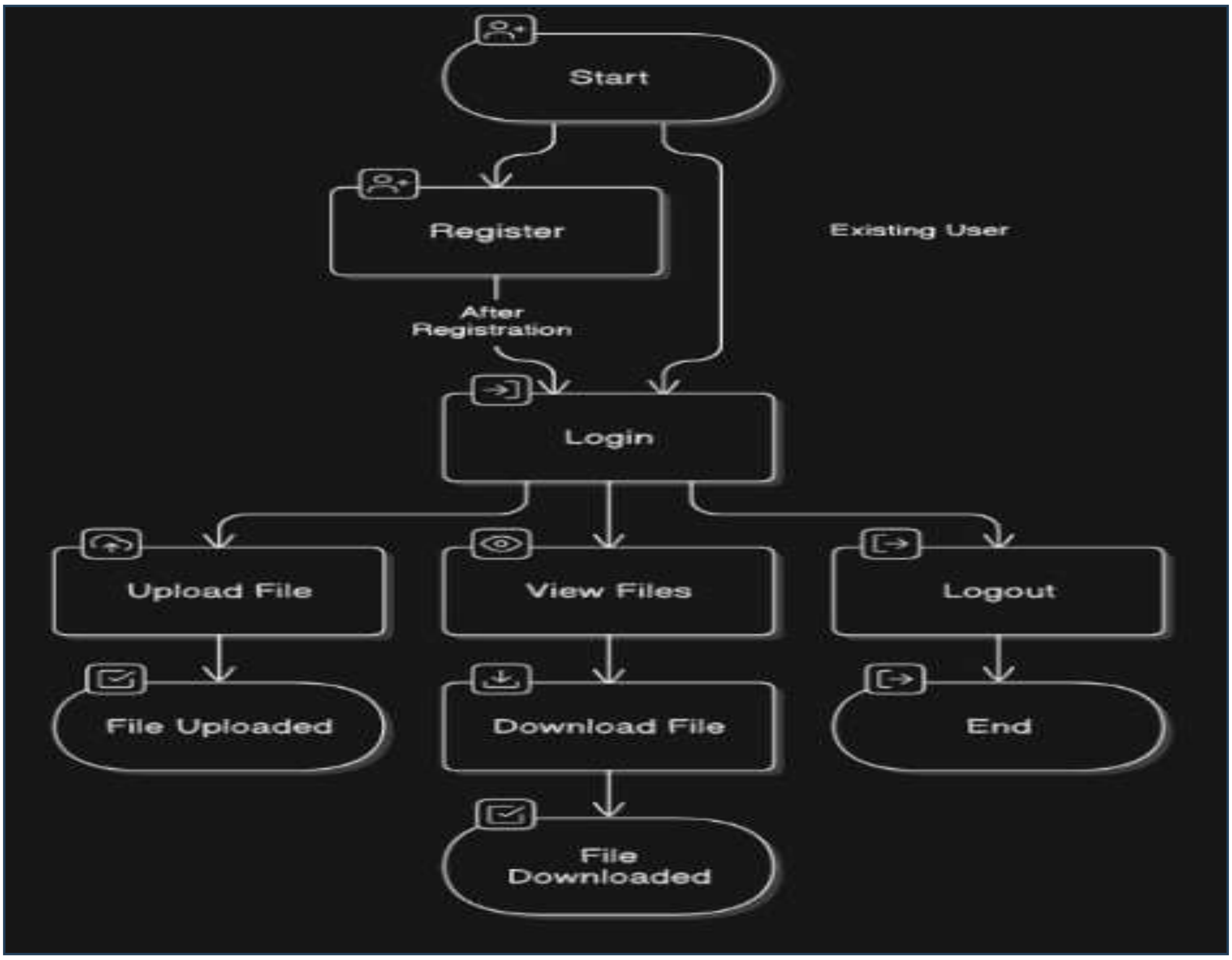


Fig.4: Use Case Diagram

The image is a flowchart that illustrates the process of user interaction with a file management system. It starts with the user registering or logging in, followed by options to upload files, view files, download files, or log out.

The flowchart is structured with clear steps and decision points, showing the sequence of actions a user can take, including registration, login, file upload, file download, and logout.

This visual representation makes it easier to understand the user journey in a file management system and helps identify any potential improvements or issues in the workflow.

4.5 SEQUENCE DIAGRAM

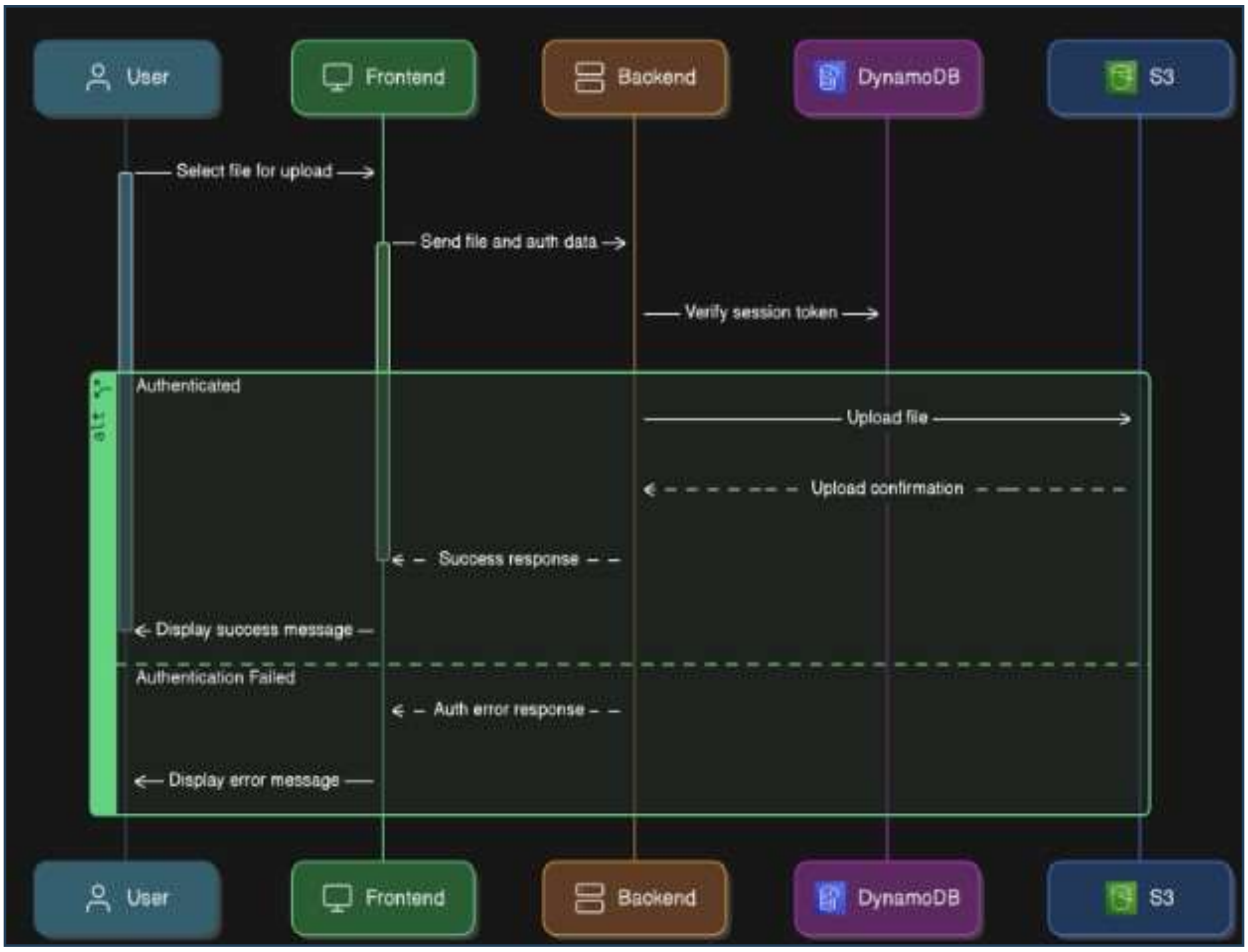


Fig.5: Sequence Diagram

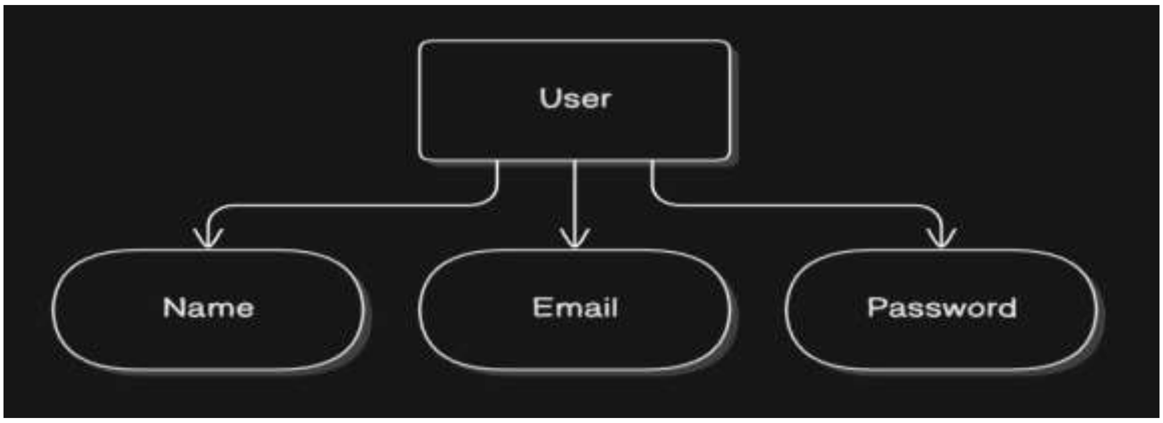
Sequence Diagram: The image depicts a sequence diagram illustrating the process of uploading a file. It shows interactions between different components: the user, frontend, backend, DynamoDB, and S3 services.

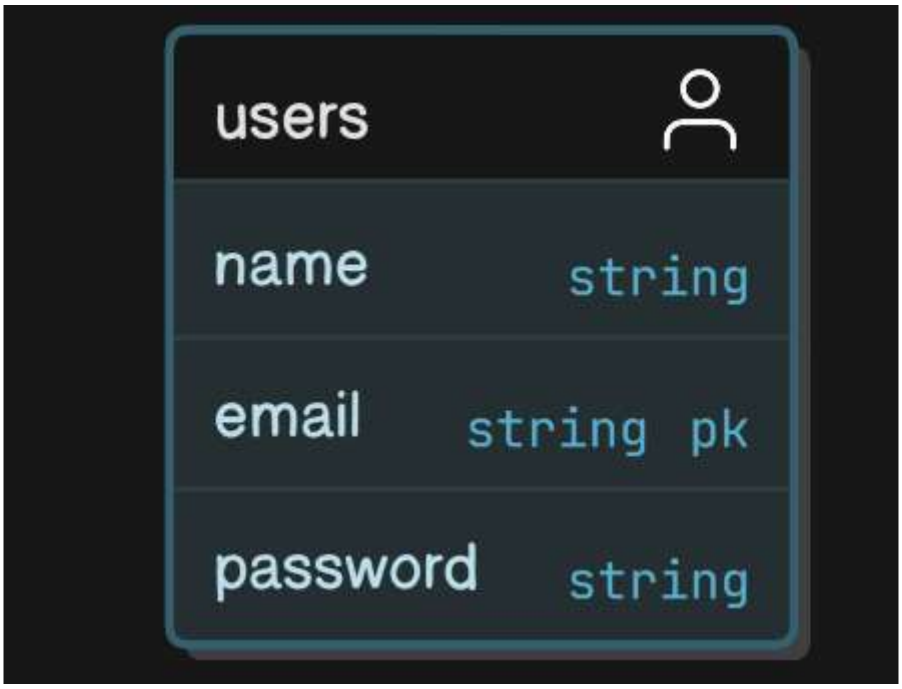
User Interaction: The process starts with the user selecting a file to upload. The frontend sends this file, along with the user's session token, to the backend for processing.

Upload Process: The user selects a file for upload, which is then sent along with authentication data to the backend. The backend verifies the session token with DynamoDB and, upon successful authentication, uploads the file to S3.

Response Handling: Depending on the outcome, the backend sends either a successful response or an authentication error response back to the frontend, which then displays the appropriate message to the user.

4.6 ENTITY-RELATIONSHIP DIAGRAM





Entity:

- The rectangle labeled User represents the entity in this diagram.
- An entity corresponds to an object or concept in a database, which is stored as a record or table.

Attributes:

- The ovals below the "User" entity represent its attributes:
  - Name: A field to store the user's name.
  - Email: A field to store the user's email address.
  - Password: A field to store the user's password.
- These attributes are connected to the "User" entity by lines, indicating that they belong to the "User."

Relationships:

- In this case, the relationships between the entity and its attributes are simple: the attributes describe or define properties of the "User."

4.7 ACTIVITY DIAGRAM

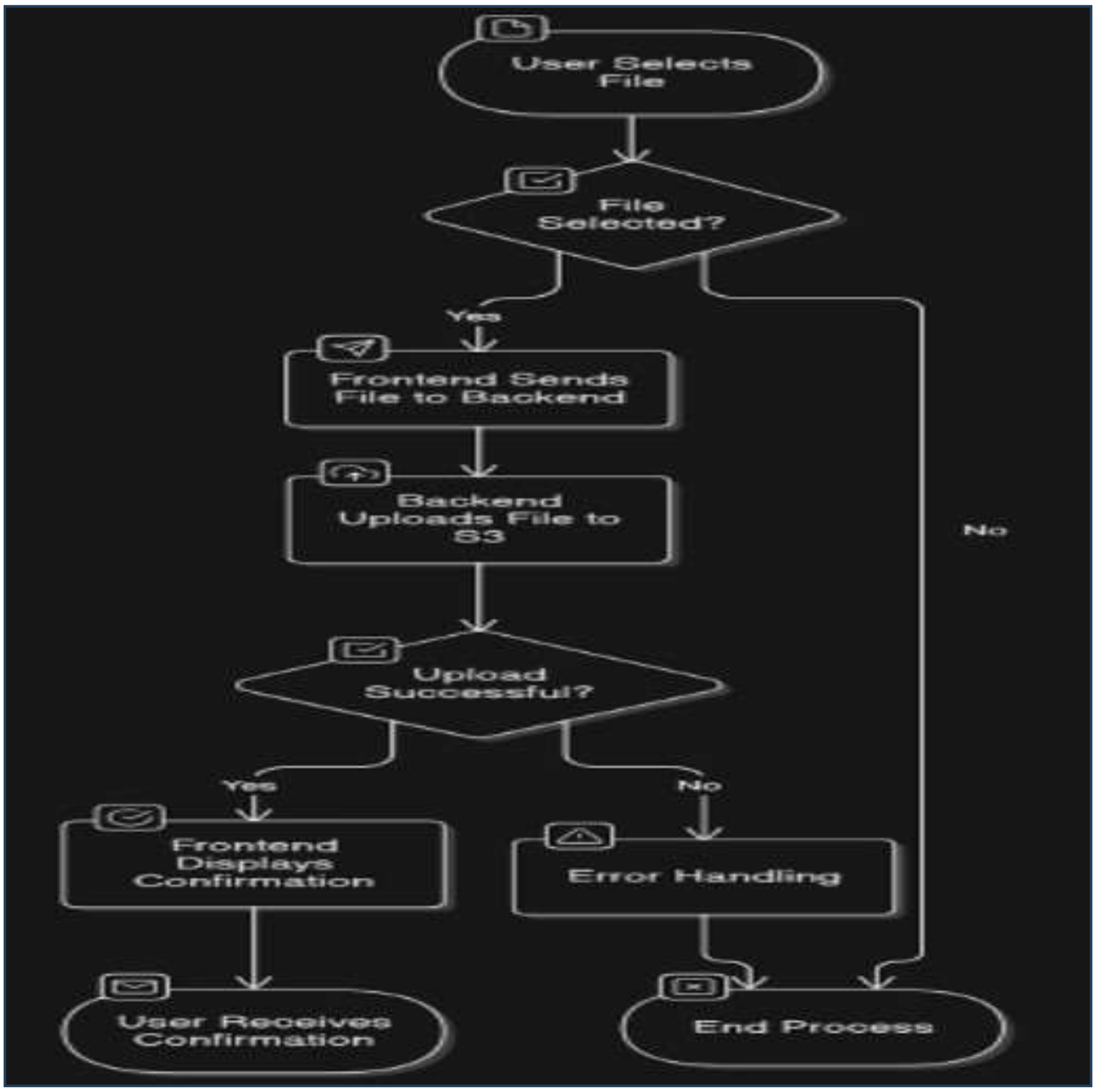

User Selects File
File Selected?
Yes
No
Frontend Sends File to Backend
Backend Uploads File to S3
Upload Successful?
Yes
No
Frontend Displays Confirmation
Error Handling
User Receives Confirmation
End Process

Fig.6: Activity Diagram

User Initiation: The process begins with the user selecting a file to upload.

Frontend Action: Once a file is selected, the frontend sends the file to the backend system.

Backend Processing: The backend uploads the file to the S3 storage service. The flowchart checks if the upload is successful or not.

Response Handling: If the upload is successful, the front end displays a confirmation message to the user. If the upload fails, error handling occurs, and the process ends with an error message.

4.8 INPUT/OUTPUT INTERFACE DESIGN

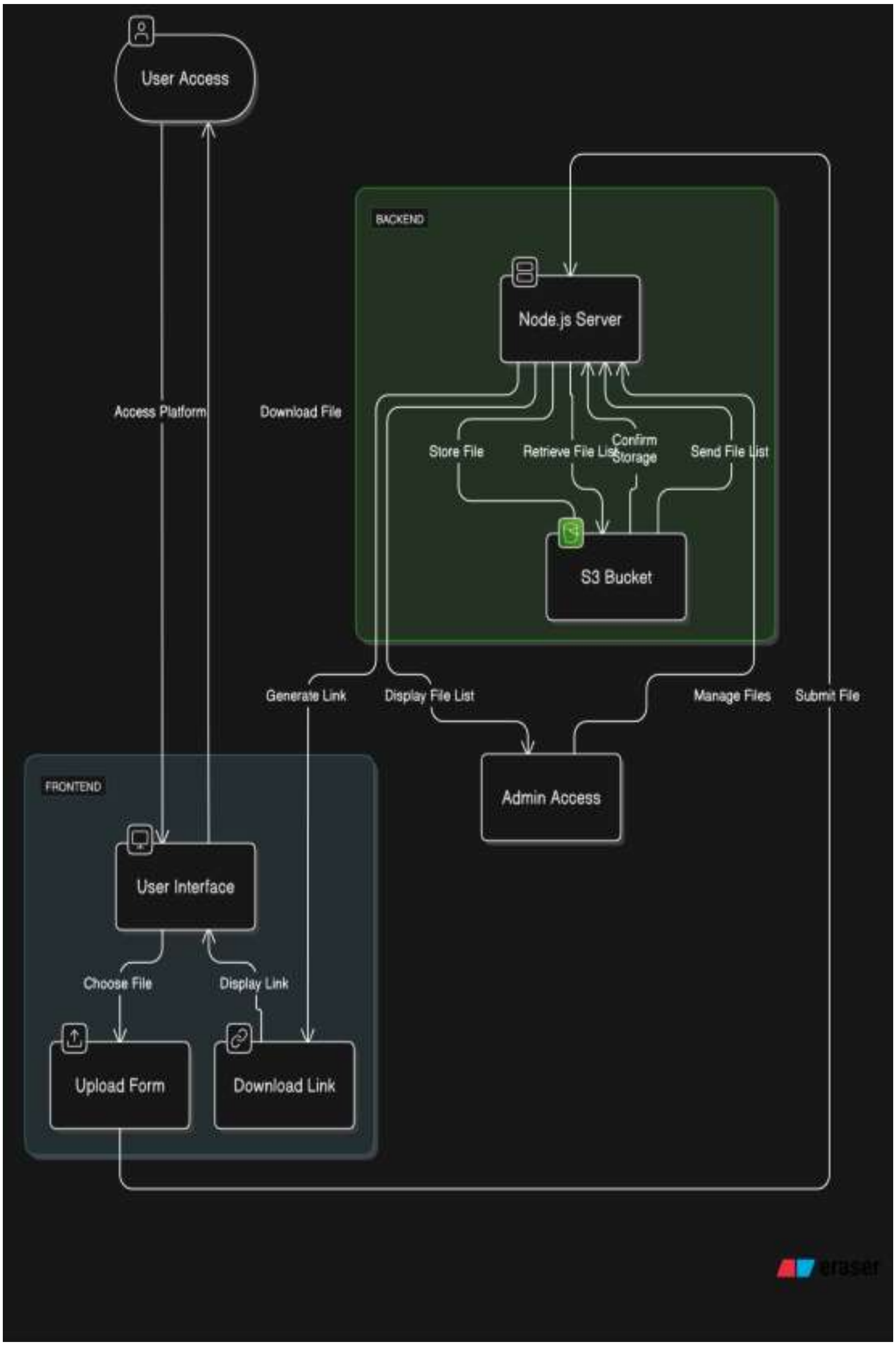


Fig.7: Input/Output Diagram

4.9 MODULE DESCRIPTION

User Authentication Module:

- **Technology Used:** Node.js, HTML, CSS, JavaScript
- **Functionality:** This module handles user registration, login, and session management. It ensures that only authenticated users can access the file-sharing features. The frontend provides forms for user input, while the backend verifies credentials and manages session tokens.

File Upload Module:

- **Technology Used:** HTML, CSS, JavaScript, Amazon S3
- **Functionality:** This module allows users to upload files to the platform. The front end includes an intuitive interface for selecting and uploading files. The backend processes the uploads, interacts with Amazon S3 to store the files securely, and provides feedback on the upload status.

File Storage Module:

- **Technology Used:** Amazon S3, Node.js
- **Functionality:** This module manages the storage of files on Amazon S3. It handles operations such as storing uploaded files, retrieving files for download, and deleting files. The backend interacts with Amazon S3 using the AWS SDK to perform these operations.

File Management Module:

- **Technology Used:** HTML, CSS, JavaScript, Node.js
- **Functionality:** This module provides users with a dashboard to view and manage their uploaded files. Users can see a list of their files, view file metadata, and perform actions such as downloading or deleting files. The frontend displays this information, while the backend handles the file operations with Amazon S3.

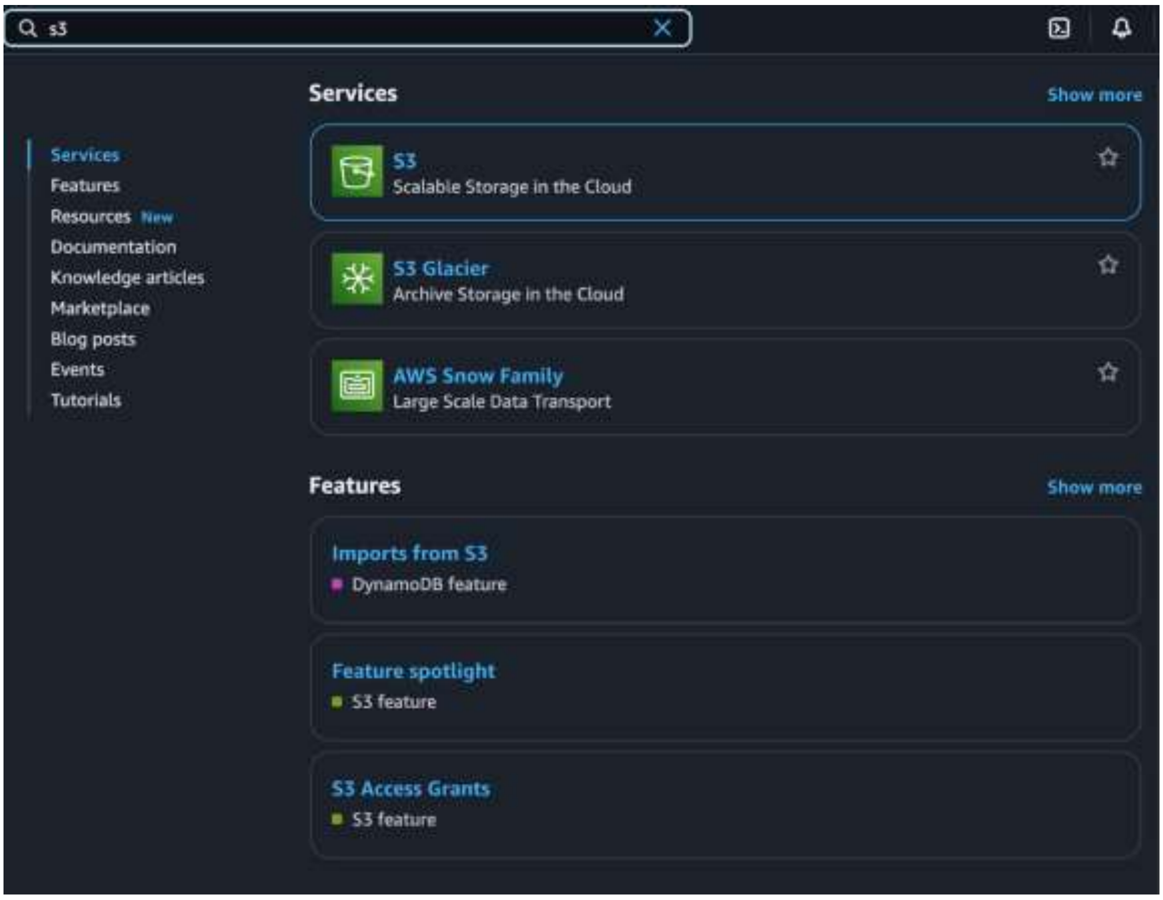
Error Handling and Notifications Module:

- **Technology Used:** JavaScript, Node.js
- **Functionality:** This module ensures robust error handling and user notifications. It captures any errors that occur during file operations, authentication, or other processes, and provides meaningful feedback to the user. The frontend displays error messages or success notifications, while the backend logs errors for further analysis.

Steps to create S3 bucket in AWS

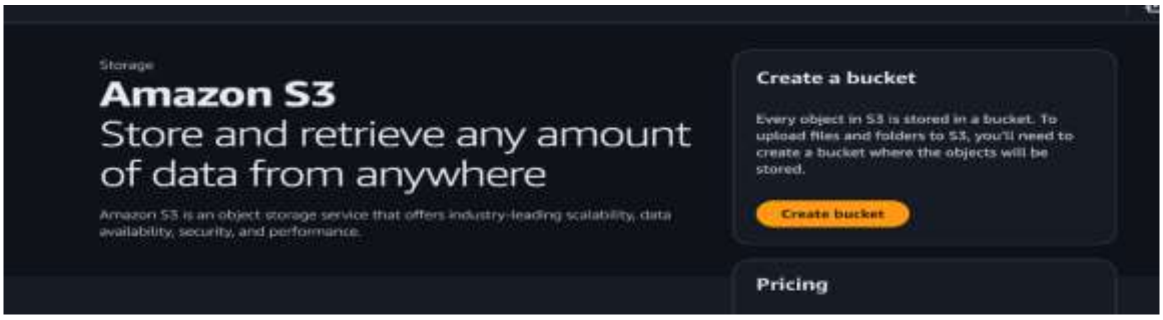
Step 1: Navigate to the S3 Service

1. On the AWS Console homepage, use the Search bar at the top and type "S3."
2. Select S3 from the list of services.



Step 2: Create a Bucket

1. On the S3 dashboard, click the "Create bucket" button.



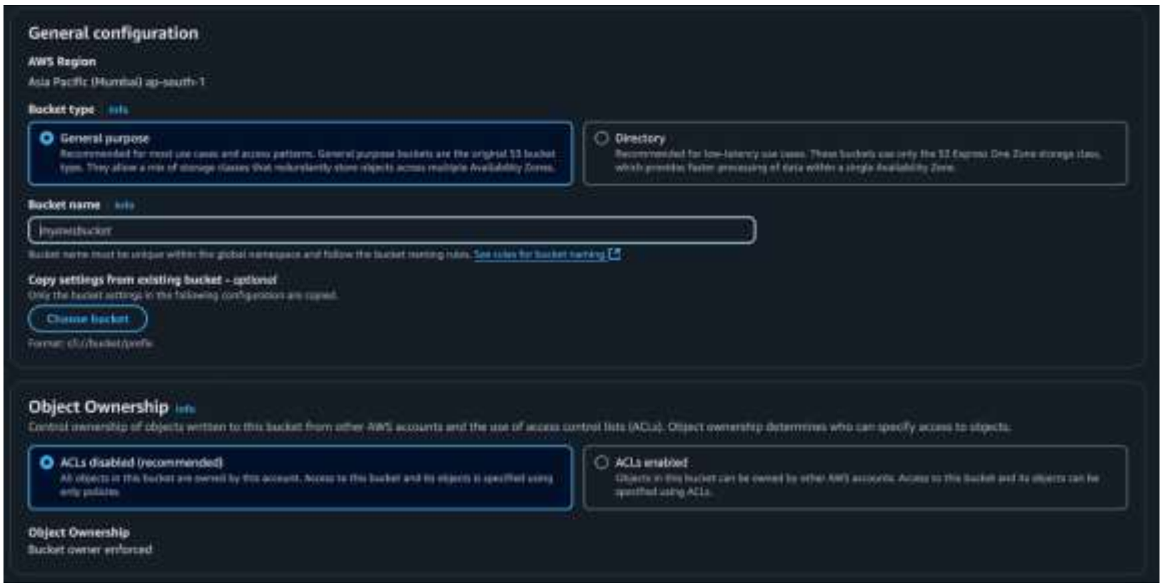
Amazon S3
Store and retrieve any amount of data from anywhere

Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance.

Every object in S3 is stored in a bucket. To upload files and folders to S3, you'll need to create a bucket where the objects will be stored.

Step 3: Configure Bucket Details

1. Enter a Bucket Name (e.g., instanote-bucket).
    - The name must be unique across all S3 buckets globally.



General configuration

AWS Region
Asia Pacific (Mumbai) ap-south-1

Bucket type Info

[✓ General purpose] Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory: Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info
myawsbucket
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
Choose bucket
Format: s3://bucket/prefix

Object Ownership Info
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

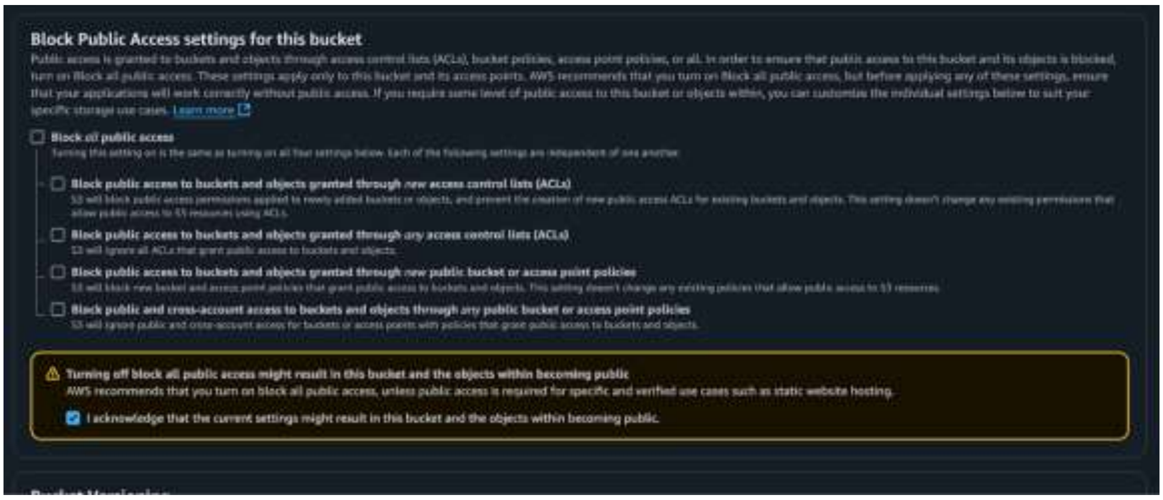
[✓ ACLs disabled (recommended)] All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

ACLs enabled: Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership
Bucket owner enforced

Step 4: Set Permissions

1. Uncheck the "Block all public access" option if you want to allow public access to the bucket. Otherwise, leave it checked.
2. Review the permission settings.



Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

☐ Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

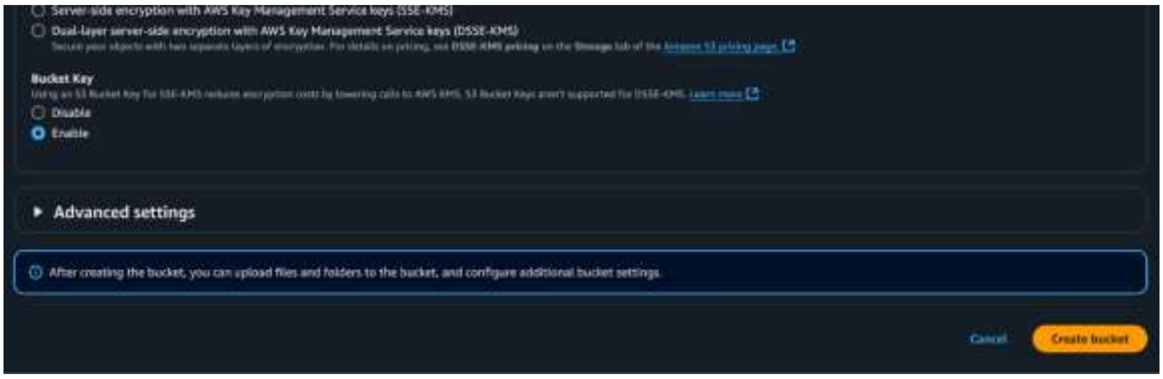
- ☐ Block public access to buckets and objects granted through *new* access control lists (ACLs)
  S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ Block public access to buckets and objects granted through *any* access control lists (ACLs)
  S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ Block public access to buckets and objects granted through *new* public bucket or access point policies
  S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ Block public and cross-account access to buckets and objects through *any* public bucket or access point policies
  S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

⚠ Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☑ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Step 5: Create the Bucket

1. Review the settings.
2. Click the "Create bucket" button.



○ Server-side encryption with AWS Key Management Service keys (SSE-KMS)

○ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)
Secure your objects with two separate layers of encryption. For details on pricing, see DSSE-KMS pricing on the Storage tab of the Amazon S3 pricing page

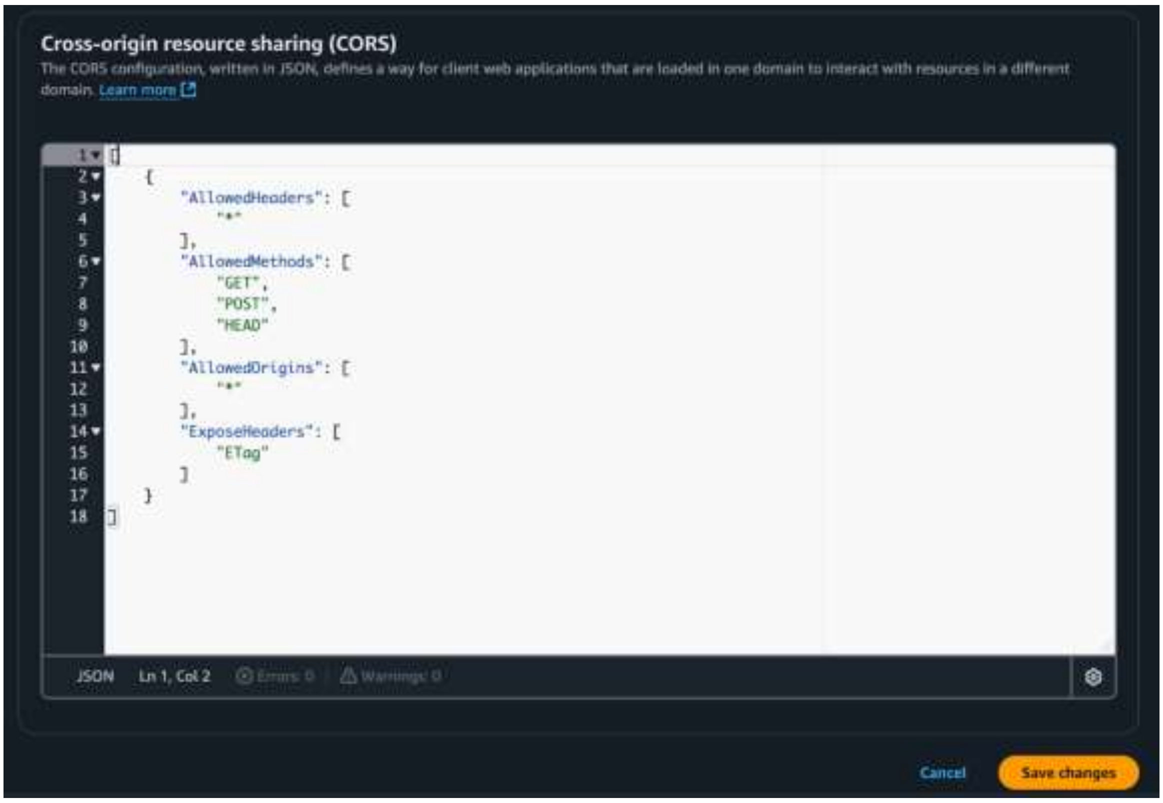
Bucket Key
Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. Learn more

Disable [✓ Enable]

After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

Step 6: Edit CORS Configuration

1. elect the Permissions tab from the bucket options.
2. Scroll down to the Cross-origin resource sharing (CORS) section.
3. Click Edit and enter a CORS configuration in JSON format and click "Save changes".



Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. Learn more

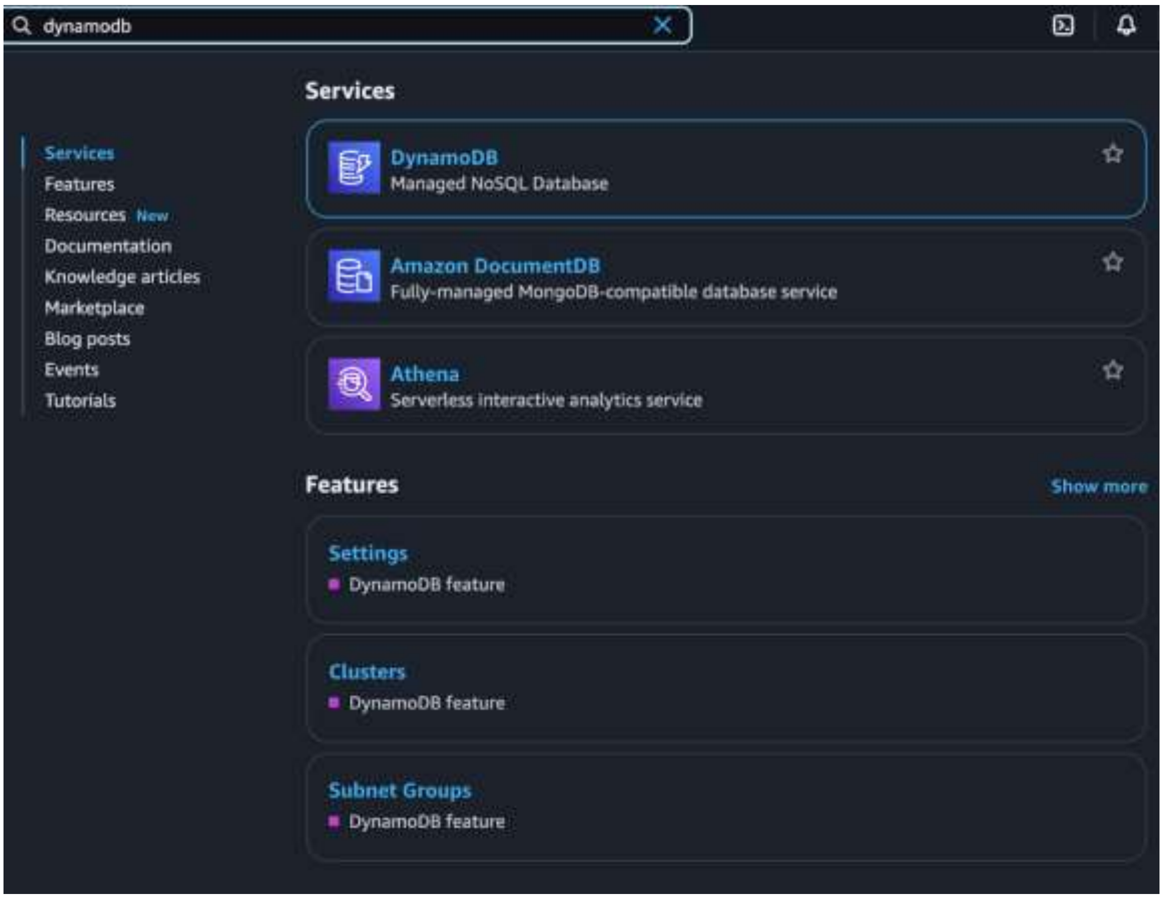
```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "GET",
            "POST",
            "HEAD"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "ETag"
        ]
    }
]
```

JSON Ln 1, Col 2 Errors: 0 Warnings: 0

Creating DynamoDB in AWS

Step 1: Navigate to DynamoDB

1. On the AWS Console homepage, use the Search bar at the top and type "DynamoDB."
2. Select DynamoDB from the list of services.



Services

DynamoDB
Managed NoSQL Database

Amazon DocumentDB
Fully-managed MongoDB-compatible database service

Athena
Serverless interactive analytics service

Features

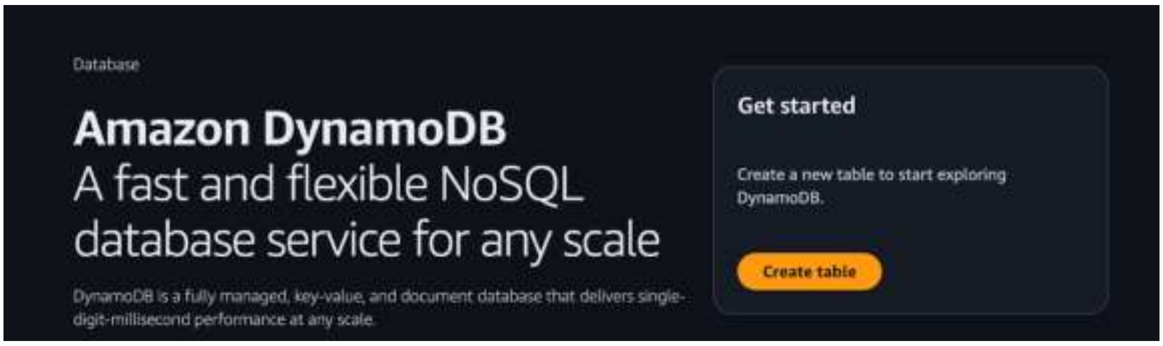
Settings
DynamoDB feature

Clusters
DynamoDB feature

Subnet Groups
DynamoDB feature

Step 2: Create a DynamoDB Table

1. On the DynamoDB dashboard, click the "Create table" button.



Database

Amazon DynamoDB
A fast and flexible NoSQL database service for any scale

DynamoDB is a fully managed, key-value, and document database that delivers single-digit-millisecond performance at any scale.

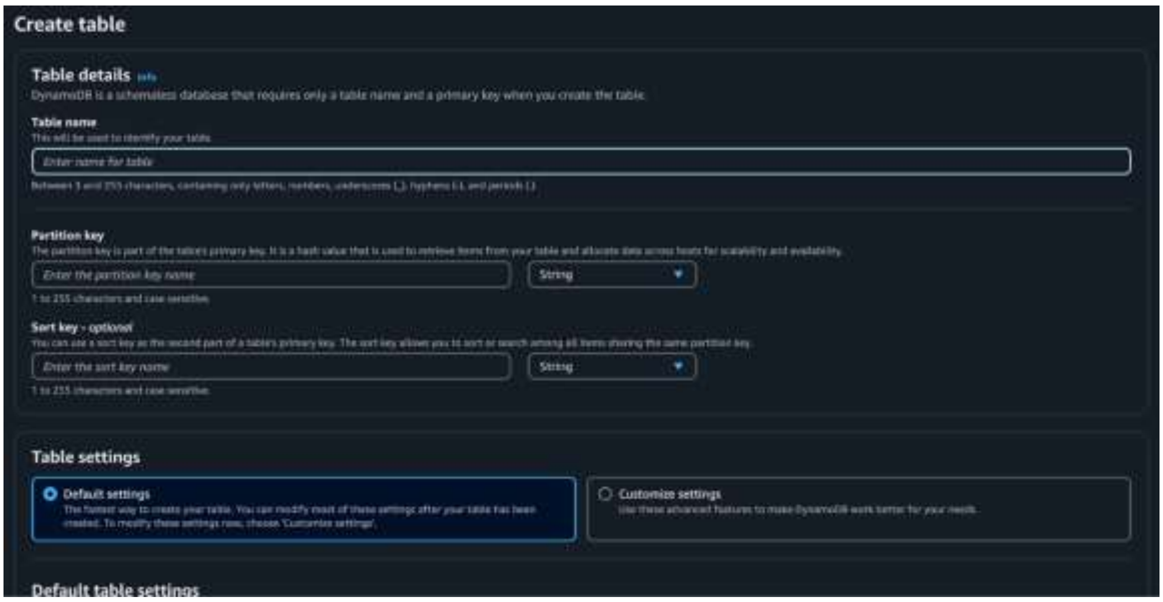
Get started

Create a new table to start exploring DynamoDB.

Step 3: Configure Table Details

1. Table name: Enter a unique name for your table (e.g. users).
2. Partition key: Enter a primary key (e.g. email) and select its data type (e.g., String or Number).

3.Optionally, add a Sort key for composite primary keys.



Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Enter name for table

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

Enter the partition key name | String

1 to 255 characters and case sensitive.

Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name | String

1 to 255 characters and case sensitive.

Table settings

[✓ Default settings] The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

Customize settings: Use these advanced features to make DynamoDB work better for your needs.

Default table settings

Step 4: Review and Create Table

1. Review all the table settings.
2. Click the "Create table" button.

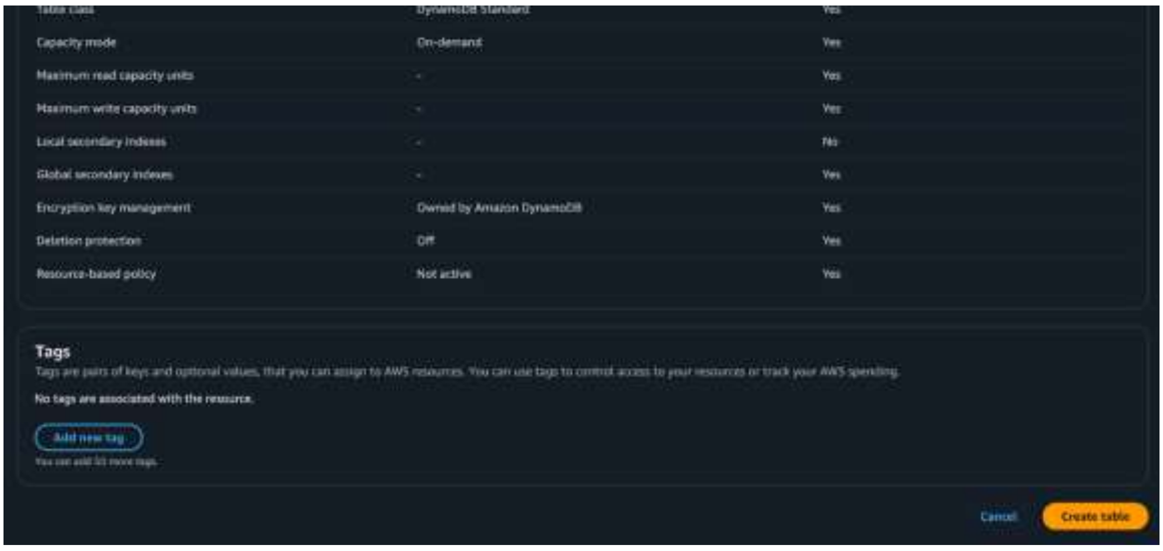


Table class	DynamoDB Standard	Yes
Capacity mode	On-demand	Yes
Maximum read capacity units	-	Yes
Maximum write capacity units	-	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

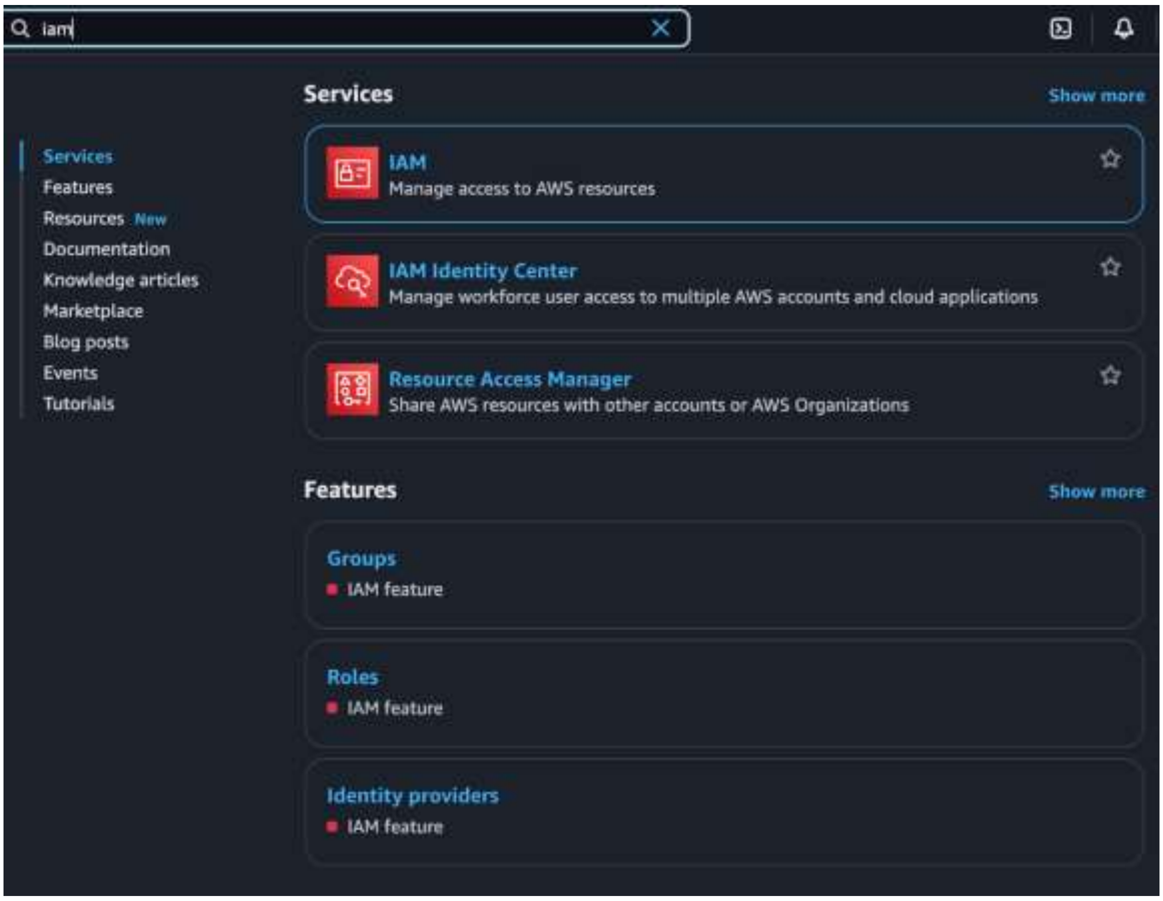
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Creating IAM users in AWS

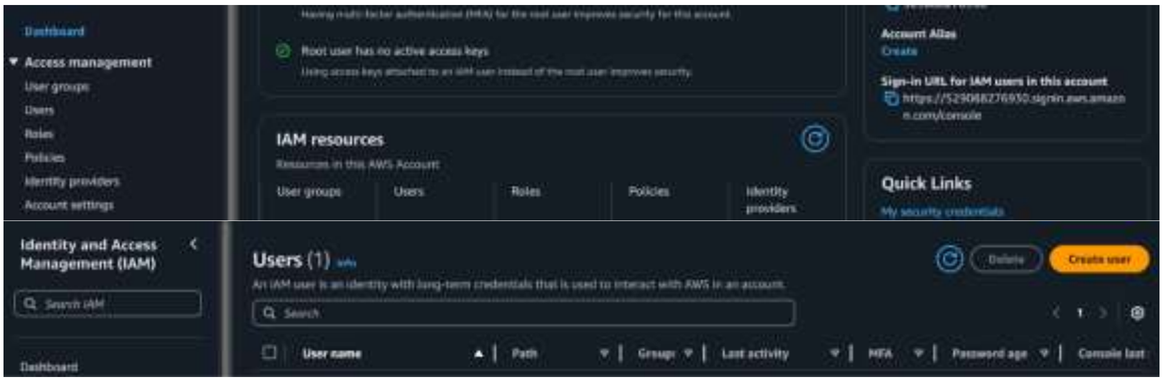
Step 1: Navigate to IAM

1. On the AWS Console homepage, use the Search bar at the top and type "IAM."
2. Select IAM from the list of services.



Step 2: Add a New User

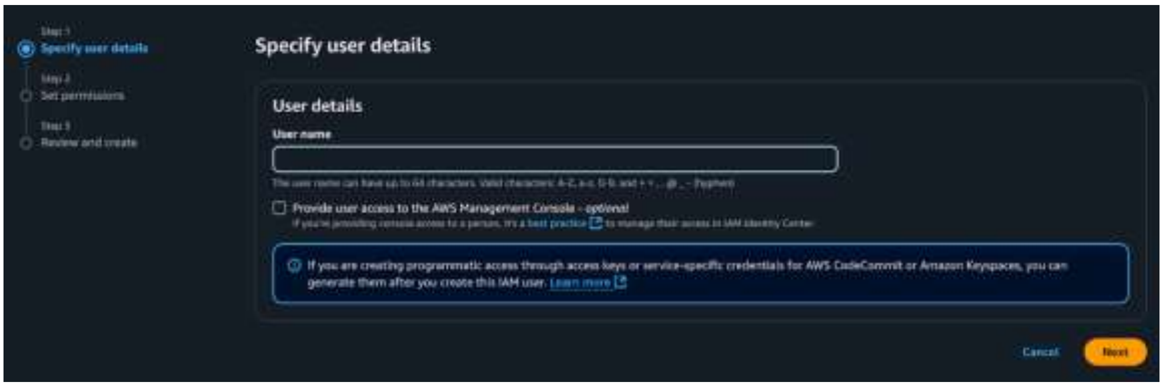
1. On the IAM dashboard, click the "Users" section in the left-hand menu.
2. Click the "Create user" button.



An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Step 3: Configure User Details

1. Enter a Username (e.g., s3-dynamodb-user).
2. Click “Next”.



Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create

Specify user details

User details

User name

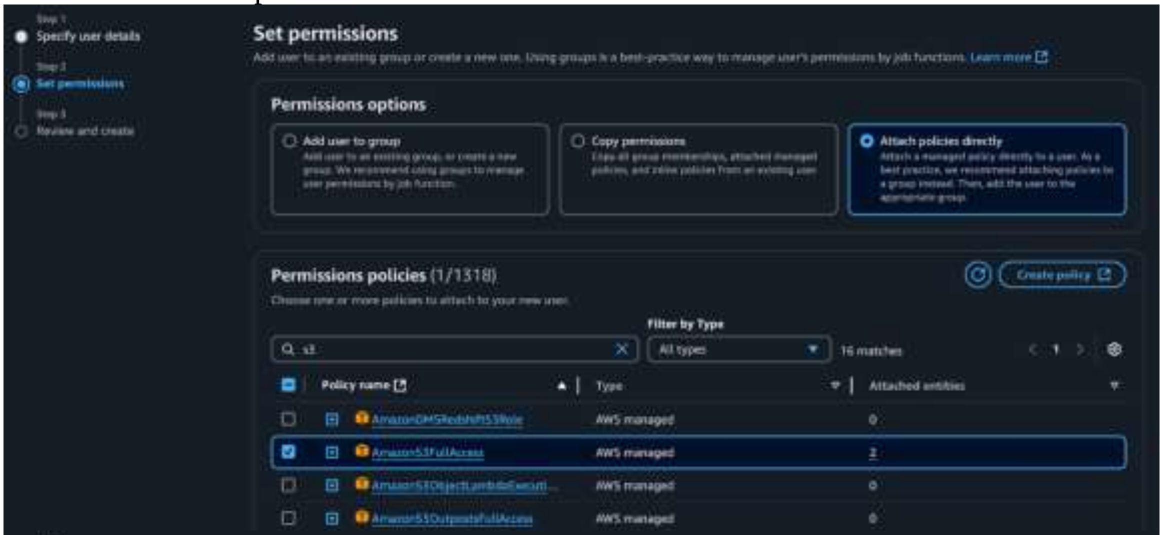
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - optional
If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

Step 4: Attach Permissions

1. Choose "Attach policies directly".
2. Search for the following managed policies:
    - AmazonS3FullAccess: Grants full access to S3.
    - Amazon DynamoDB Full Access: Grants full access to DynamoDB.
3. Check both policies.



Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

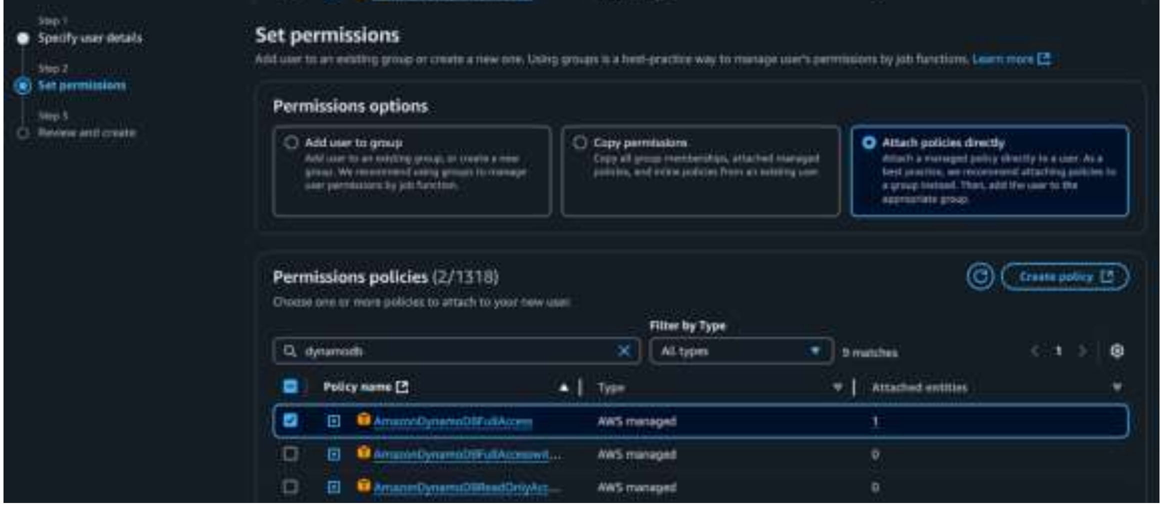
● Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1318)

Choose one or more policies to attach to your new user.

Search: s3 | Filter by Type: All types | 16 matches

	Policy name	Type	Attached entities
☐	AmazonDMSRedshiftS3Role	AWS managed	0
☑	AmazonS3FullAccess	AWS managed	2
☐	AmazonS3ObjectLambdaExecuti...	AWS managed	0
☐	AmazonS3OutpostsFullAccess	AWS managed	0



Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

● Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (2/1318)

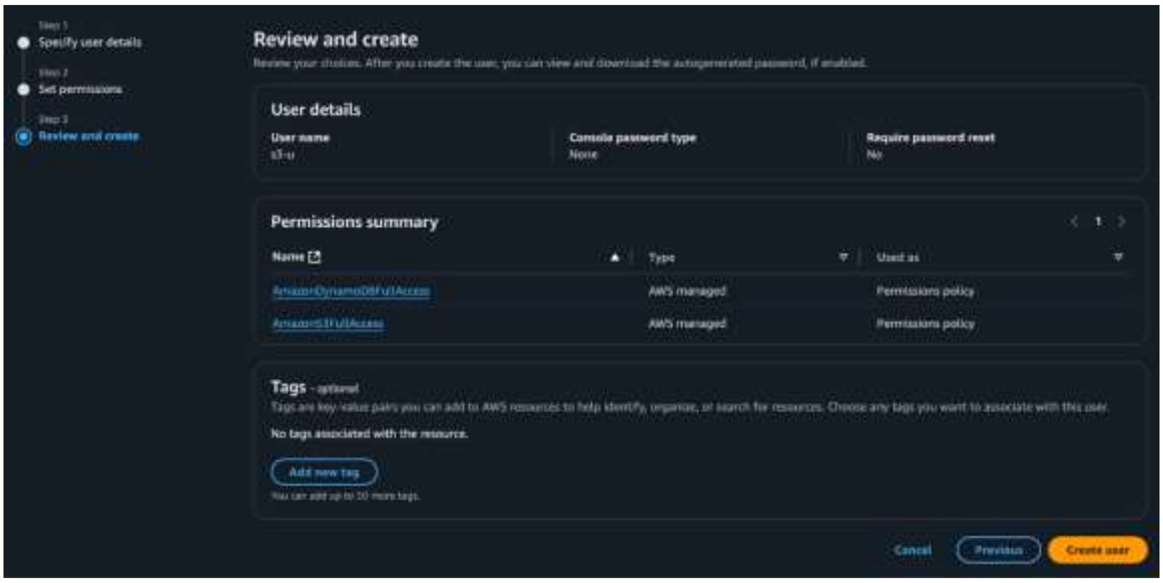
Choose one or more policies to attach to your new user.

Search: dynamodb | Filter by Type: All types | 9 matches

	Policy name	Type	Attached entities
☑	AmazonDynamoDBFullAccess	AWS managed	1
☐	AmazonDynamoDBFullAccesswit...	AWS managed	0
☐	AmazonDynamoDBReadOnlyAcc...	AWS managed	0

Step 5: Review and Create User

1. Review the user details, permissions, and settings.
2. Click the "Create user" button.



Review and create

Review your choices. After you create the user, you can view and download the autogenerated password, if enabled.

User details

User name	Console password type	Require password reset
s3-u	None	No

Permissions summary

Name	Type	Used as
AmazonDynamoDBFullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy

Tags - optional

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

You can add up to 50 more tags.

5. CODING AND IMPLEMENTATION

1. Install Node.js: Ensure that Node.js is installed on your machine. You can download it from Node.js official site.
2. Initialize a New Project:
   - Open your terminal and navigate to the folder where you want to create the project.
   - Run the following command to initialize a package.json file:

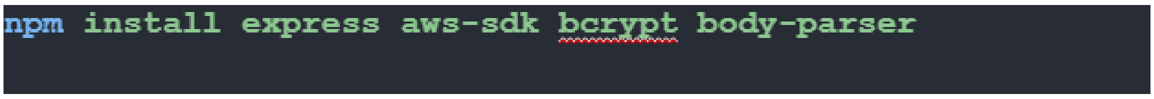
```
npm init -y
```

3. Set Up the Folder Structure

```
instanote/
├── node_modules/
├── public/
|    ├── dashboard.css/
|    ├── dashboard.html/
|    ├── dashboard.js/
|    ├── login.css/
|    ├── login.html/
|    └── login.js
└── package.json
```

4. Install Dependencies

- Install the necessary dependencies for your project:



```
npm install express aws-sdk bcrypt body-parser
```

5. Write the code for the files:

app.js

```
// Import necessary modules
const express = require("express");
const AWS = require("aws-sdk");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const bodyParser = require("body-parser");

const app = express();

app.use((req, res, next) => {
 res.setHeader("Access-Control-Allow-Origin", "*");
 res.setHeader("Access-Control-Expose-Headers", "ETag");
 next();
});

app.use(express.static(__dirname + "/public"));

// Serve login.html as the default file for "/"
app.get("/", (req, res) => {
 res.sendFile(__dirname + '/public/login.html');
});

const port = 3000;
const SECRET_KEY =
 "d8aab0cbcd42942210c19e312f9525c1804f87031a8e505d6c355fd0728d3fda";          //
Replace with a secure secret key

// Configure AWS SDK
AWS.config.update({
 region: "region",
 accessKeyId: "access-key-id",
 secretAccessKey: "secret-access-key",
});

const dynamoDb = new AWS.DynamoDB.DocumentClient();
const USERS_TABLE = "Users"; // DynamoDB table name

// Middleware
app.use(bodyParser.json());
```

```
// Register a new user
app.post("/register", async (req, res) => {
 const { name, email, password } = req.body;

 if (!name || !email || !password) {
   return res
     .status(400)
     .json({ error: "Name, email, and password are required" });
 }

 try {
   // Hash the password
   const hashedPassword = await bcrypt.hash(password, 10);

   // Check if user already exists
   const existingUser = await dynamoDb
     .get({ TableName: USERS_TABLE, Key: { email } })
     .promise();

   if (existingUser.Item) {
     return res.status(400).json({ error: "User already exists" });
   }

   // Save user to DynamoDB
   const newUser = {
     email,
     name,
     password: hashedPassword,
   };

   await dynamoDb.put({ TableName: USERS_TABLE, Item: newUser }).promise();

   res.status(201).json({ message: "User registered successfully" });
 } catch (error) {
   console.error("Error registering user:", error);
   res.status(500).json({ error: "Internal server error" });
 }
});

// Sign in a user
app.post("/signin", async (req, res) => {
 const { email, password } = req.body;

 if (!email || !password) {
   return res.status(400).json({ error: "Email and password are required" });
 }

 try {
   // Fetch user from DynamoDB
   const user = await dynamoDb
     .get({ TableName: USERS_TABLE, Key: { email } })
     .promise();

   if (!user.Item) {
     return res.status(400).json({ error: "User not found" });
   }

   // Compare password
   const isPasswordValid = await bcrypt.compare(password, user.Item.password);

   if (!isPasswordValid) {
     return res.status(400).json({ error: "Invalid password" });
   }
   res.status(200).json({ message: "Login successful", token });
 } catch (error) {
   console.error("Error signing in user:", error);
   res.status(500).json({ error: "Internal server error" });
 }
});

// Start the server
app.listen(port, () => {
 console.log(`Server is running on http://localhost:${port}`);
});
```

login.html

```
<!DOCTYPE html>
<html lang="en">

<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css" />
 <link rel="stylesheet" href="/login.css" />
 <title>Login Page</title>
</head>

<body>
 <div class="container" id="container">
   <div class="form-container sign-up">
     <form id="signup-form">
       <h1>Create Account</h1>
       <input type="text" id="signup-name" placeholder="Name" required />
       <input type="email" id="signup-email" placeholder="Email" required />
       <input type="password" id="signup-password" placeholder="Password" required />
       <button>Sign Up</button>
     </form>
   </div>
   <div class="form-container sign-in">
     <form id="signin-form">
       <h1>Sign In</h1>
       <input type="email" id="signin-email" placeholder="Email" required />
       <input type="password" id="signin-password" placeholder="Password" required />
       <button type="submit">Sign In</button>
     </form>
   </div>
   <div class="toggle-container">
     <div class="toggle">
       <div class="toggle-panel toggle-left">
         <h1>Welcome Back!</h1>
         <p>Enter your personal details to use all of site features</p>
         <button class="hidden" id="login">Sign In</button>
       </div>
       <div class="toggle-panel toggle-right">
         <h1>Hello, Friend!</h1>
         <p>
           Register with your personal details to use all of site features
         </p>
         <button class="hidden" id="register" type="submit">Sign Up</button>
       </div>
     </div>
   </div>
 </div>
 <script src="/login.js"></script>
</body>

</html>
```

login.css

```
@import
url("https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500;600;700&
display=swap");

* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
 font-family: "Montserrat", sans-serif;
}

body {
 background-color: #c9d6ff;
 background: linear-gradient(to right, #e2e2e2, #c9d6ff);
 display: flex;
 align-items: center;
 justify-content: center;
 flex-direction: column;
 height: 100vh;
}

.container {
 background-color: #fff;
 border-radius: 30px;
 box-shadow: 0 5px 15px rgba(0, 0, 0, 0.35);
 position: relative;
 overflow: hidden;
 width: 768px;
 max-width: 100%;
 min-height: 480px;
}

.container p {
 font-size: 14px;
 line-height: 20px;
 letter-spacing: 0.3px;
 margin: 20px 0;
}

.container span {
 font-size: 12px;
}

.container a {
 color: #333;
 font-size: 13px;
 text-decoration: none;
 margin: 15px 0 10px;
}

.container button {
 background-color: #213555;
 color: #fff;
 font-size: 12px;
 padding: 10px 45px;
 border: 1px solid transparent;
 border-radius: 8px;
 font-weight: 600;
 letter-spacing: 0.5px;
 text-transform: uppercase;
 margin-top: 10px;
 cursor: pointer;
}

.container button.hidden {
 background-color: transparent;
 border-color: #fff;
}
```

```
.container form {
 background-color: #fff;
 display: flex;
 align-items: center;
 justify-content: center;
 flex-direction: column;
 padding: 0 40px;
 height: 100%;
}

.container input {
 background-color: #eee;
 border: none;
 margin: 8px 0;
 padding: 10px 15px;
 font-size: 13px;
 border-radius: 8px;
 width: 100%;
 outline: none;
}

.form-container {
 position: absolute;
 top: 0;
 height: 100%;
 transition: all 0.6s ease-in-out;
}

.sign-in {
 left: 0;
 width: 50%;
 z-index: 2;
}

.container.active .sign-in {
 transform: translateX(100%);
}

.sign-up {
 left: 0;
 width: 50%;
 opacity: 0;
 z-index: 1;
}

.container.active .sign-up {
 transform: translateX(100%);
 opacity: 1;
 z-index: 5;
 animation: move 0.6s;
```

```
}

@keyframes move {
 0%,
 49.99% {
   opacity: 0;
   z-index: 1;
 }
 50%,
 100% {
   opacity: 1;
   z-index: 5;
 }
}

.toggle-container {
 position: absolute;
 top: 0;
 left: 50%;
 width: 50%;
 height: 100%;
 overflow: hidden;
 transition: all 0.6s ease-in-out;
 border-radius: 150px 0 0 100px;
 z-index: 1000;
}

.container.active .toggle-container {
 transform: translateX(-100%);
 border-radius: 0 150px 100px 0;
}

.toggle {
 background-color: #512da8;
 height: 100%;
 background: linear-gradient(to right, #213555, #213555);
 color: #fff;
 position: relative;
 left: -100%;
 height: 100%;
 width: 200%;
 transform: translateX(0);
 transition: all 0.6s ease-in-out;
}

.container.active .toggle {
 transform: translateX(50%);
}

.toggle-panel {
 position: absolute;
 width: 50%;
 height: 100%;
 display: flex;
 align-items: center;
 justify-content: center;
 flex-direction: column;
 padding: 0 30px;
 text-align: center;
 top: 0;
 transform: translateX(0);
 transition: all 0.6s ease-in-out;
}

.toggle-left {
 transform: translateX(-200%);
}

.container.active .toggle-left {
 transform: translateX(0);
}

.toggle-right {
 right: 0;
 transform: translateX(0);
}

.container.active .toggle-right {
 transform: translateX(200%);
}
```

login.js

```
const container = document.getElementById("container");
const registerBtn = document.getElementById("register");
const loginBtn = document.getElementById("login");

registerBtn.addEventListener("click", () => {
 container.classList.add("active");
});

loginBtn.addEventListener("click", () => {
 container.classList.remove("active");
});

// API Base URL
const API_BASE_URL = "http://localhost:3000";

// Handle Sign-Up Form Submission
document
```

```
 .getElementById("signup-form")
 .addEventListener("submit", async (event) => {
   event.preventDefault();

   const name = document.getElementById("signup-name").value;
   const email = document.getElementById("signup-email").value;
   const password = document.getElementById("signup-password").value;

   try {
     const response = await fetch(`${API_BASE_URL}/register`, {
       method: "POST",
       headers: {
         "Content-Type": "application/json",
       },
       body: JSON.stringify({ name, email, password }),
     });

     const data = await response.json();

     if (response.ok) {
       alert("Registration successful! Please sign in.");
     } else {
       alert(`Error: ${data.error}`);
     }
   } catch (error) {
     console.error("Error:", error);
     alert("An error occurred. Please try again.");
   }
 });

// Handle Sign-In Form Submission
document
 .getElementById("signin-form")
 .addEventListener("submit", async (event) => {
   event.preventDefault();

   const email = document.getElementById("signin-email").value;
   const password = document.getElementById("signin-password").value;

   try {
     const response = await fetch(`${API_BASE_URL}/signin`, {
       method: "POST",
       headers: {
         "Content-Type": "application/json",
       },
       body: JSON.stringify({ email, password }),
     });

     const data = await response.json();
```

```
    if (response.ok) {
      alert("Login successful!");
      window.location.href = "dashboard.html";
    } else {
      alert(`Error: ${data.error}`);
    }
  } catch (error) {
    console.error("Error:", error);
    alert("An error occurred. Please try again.");
  }
 });
```

dashboard.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>InstaNotes</title>
   <script src="https://sdk.amazonaws.com/js/aws-sdk-2.1488.0.min.js"></script>
   <link rel="stylesheet" href="/dashboard.css" />
 </head>

 <body>
   <div class="navbar">
     <div class="logo">IN</div>
     <ul>
       <li>
         <a href="/login.html" class="logout-link" id="logout-link">Logout</a>
       </li>
     </ul>
   </div>

   <div class="main">
     <div class="file-list" id="file-list">
       <p>Loading files...</p>
     </div>
     <div class="upload-container">
       <input type="file" id="fileInput" class="upload-btn" />
       <button class="upload-btn" id="upload-btn">Upload</button>
     </div>
   </div>

   <script src="/dashboard.js"></script>
 </body>
</html>
```

dashboard.css

```
* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
}

body {
 display: flex;
 height: 100vh;
 font-family: Arial, sans-serif;
}

.navbar,
.main
{
 padding: 20px;
 color: white;
 text-align: center;
 flex: 1;
}

.navbar {
 background-color: #213555;
 flex: 0.5;
 display: flex;
 flex-direction: column;
 align-items: center;
}

.logo {
 width: 80px;
 height: 80px;
 background-color: #ffffff;
 border-radius: 50%;
 margin-bottom: 20px;
 display: flex;
 align-items: center;
 justify-content: center;
 color: #d8c4b6;
 font-weight: bold;
 font-size: 24px;
}

.navbar ul {
 list-style-type: none;
 padding: 0;
}

.navbar ul li {
 margin: 10px 0;
}

.navbar ul li a {
 color: white;
 text-decoration: none;
 font-weight: bold;
}

.main {
 background-color: #3e5879;
 flex: 2;
 display: flex;
 flex-direction: column;
 align-items: center;
 overflow-y: auto;
}

.file-list {
 margin-top: 20px;
 width: 80%;
 background: white;
 color: black;
 border-radius: 10px;
 padding: 10px;
}

.file-item {
 padding: 10px;
 border-bottom: 1px solid #ddd;
 text-align: left;
}

.file-item a {
 color: #d8c4b6;
 text-decoration: none;
 font-weight: bold;
}
.file-item:last-child {
 border-bottom: none;
}

.upload-container {
 margin-top: 20px;
 text-align: center;
}

.upload-container input {
 margin-bottom: 10px;
}
```

```
.upload-btn {
 background-color: white;
 color: #d8c4b6;
 padding: 10px 20px;
 border: 2px solid #d8c4b6;
 border-radius: 5px;
 cursor: pointer;
 font-weight: bold;
 transition: background-color 0.3s, color 0.3s;
}

.upload-btn-label:hover {
 background-color: #d8c4b6;
 color: white;
}

.sidebar {
 background-color: #213555;
 flex: 1;
 color: #d8c4b6;
}
```

dashboard.js

```
AWS.config.update({
 accessKeyId: "access-key-id",
 secretAccessKey: "secret-access-key",
 region: "region",
});

const s3 = new AWS.S3();
const bucketName = "insta-note-store";

function listFiles() {
 const params = {
   Bucket: bucketName,
 };

 s3.listObjectsV2(params, (err, data) => {
   const fileList = document.getElementById("file-list");
   if (err) {
     console.error("Error fetching files:", err);
     fileList.innerHTML = "<p>Error loading files</p>";
   } else {
     fileList.innerHTML = "";
     data.Contents.forEach((file) => {
       const signedUrlParams = {
         Bucket: bucketName,
         Key: file.Key,
         Expires: 60,
       };

       s3.getSignedUrl("getObject", signedUrlParams, (err, url) => {
         if (err) {
           console.error("Error generating signed URL:", err);
           return;
         }
         const fileItem = document.createElement("div");
         fileItem.className = "file-item";

         fileItem.innerHTML = `<a href="${url}" download>${file.Key}</a>`;
         fileList.appendChild(fileItem);
       });
     });
   }
 });
}

const button = document.getElementById("upload-btn");
button.addEventListener("click", () => {
 const fileInput = document.getElementById("fileInput");
 button.disabled = true;
 button.textContent = "Uploading...";
 const file = fileInput.files[0];

 if (!file) {
   alert("Please select a file to upload.");
   return;
 }

 const params = {
   Bucket: bucketName,
   Key: file.name,
   Body: file,
 };

 s3.upload(params, (err, data) => {
   if (err) {
     console.error("Error uploading file:", err);
     alert("Error uploading file.");
   } else {
     alert(`File uploaded successfully: ${data.Location}`);
     button.textContent = "Upload";
     button.disabled = false; // Re-enable the button
     listFiles();
   }
 });
});
listFiles();
```

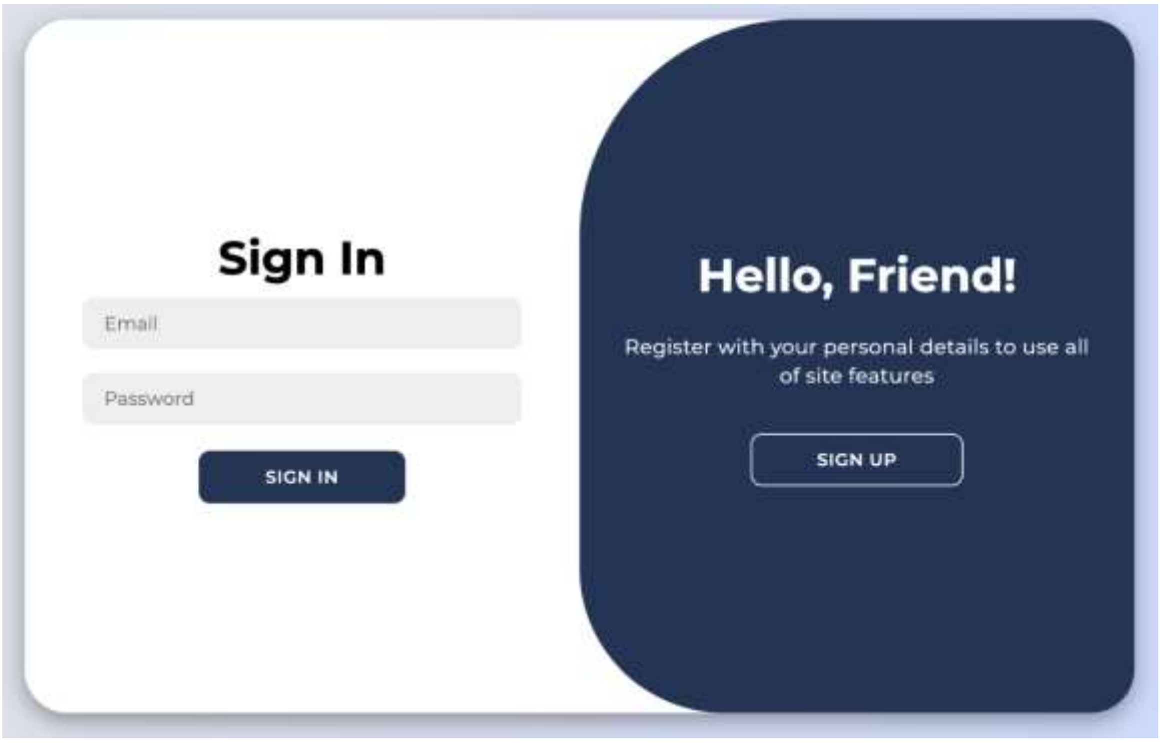
6. Run the Application

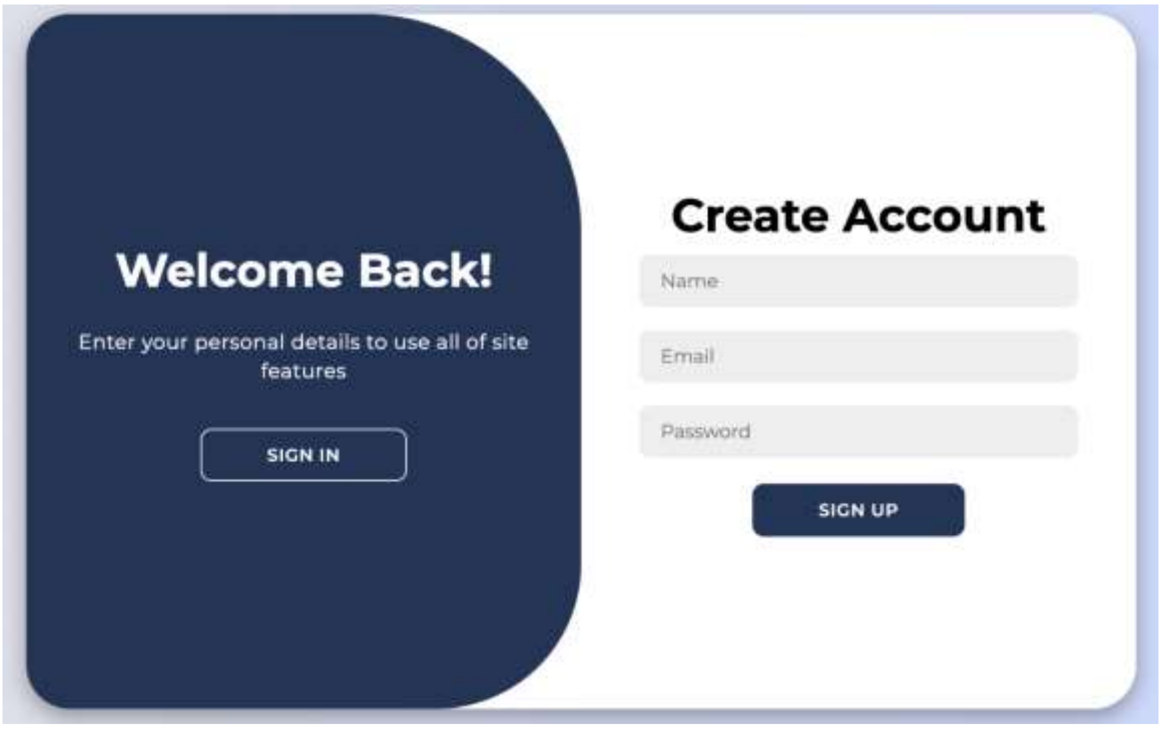
- Start the server by running:

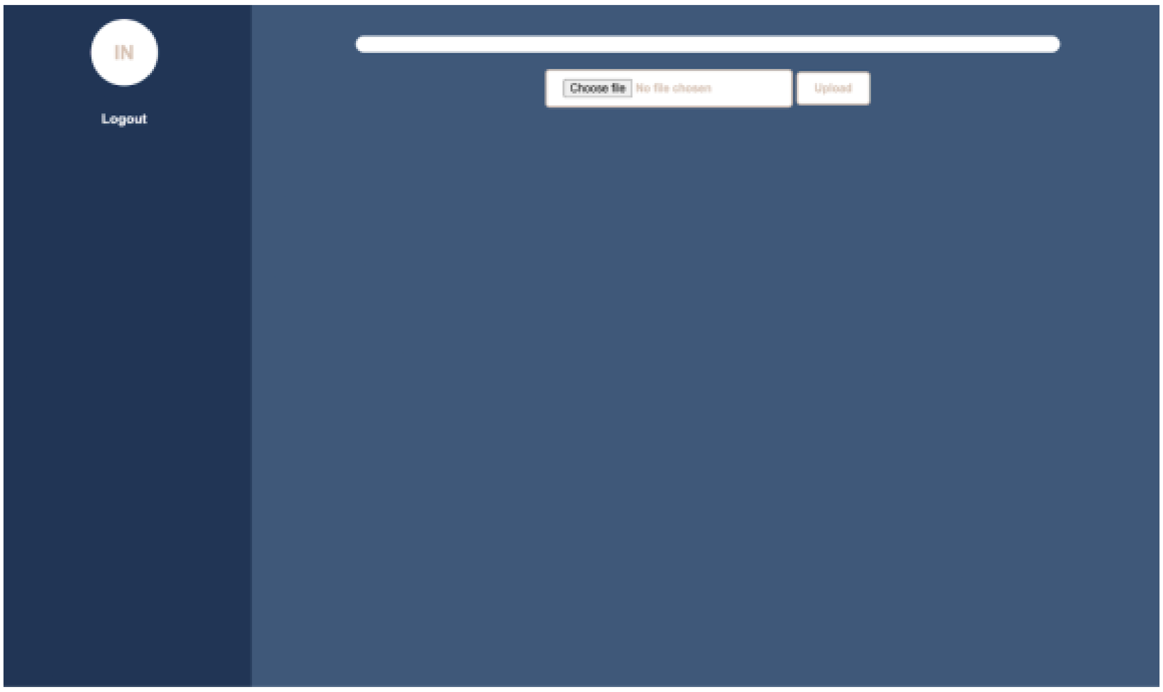
```
node app.js
```

- Open your browser and visit:

```
http://localhost:3000
```

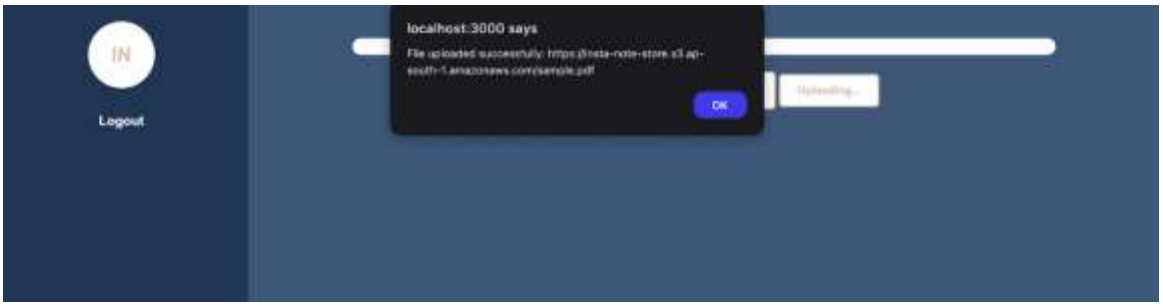

Sign In
Email
Password
SIGN IN
Hello, Friend!
Register with your personal details to use all of site features
SIGN UP





6. SOFTWARE TESTING / TEST CASES

Test 1: Uploading a PDF file.

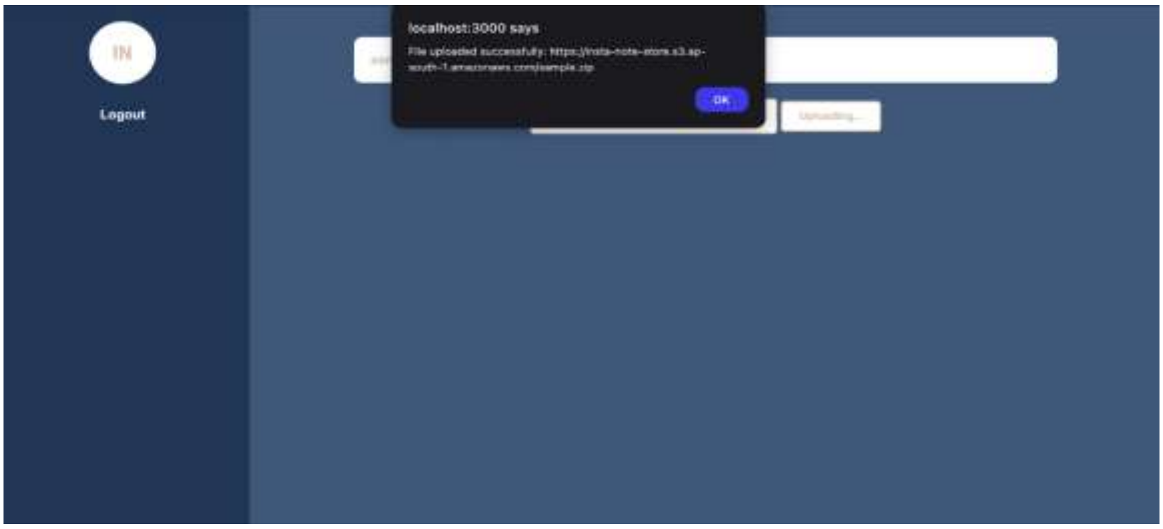


localhost:3000 says

File uploaded successfully: https://insta-note-store.s3.ap-south-1.amazonaws.com/sample.pdf

Result: Successfully uploaded a PDF file.

Test 2: Uploading a ZIP file.

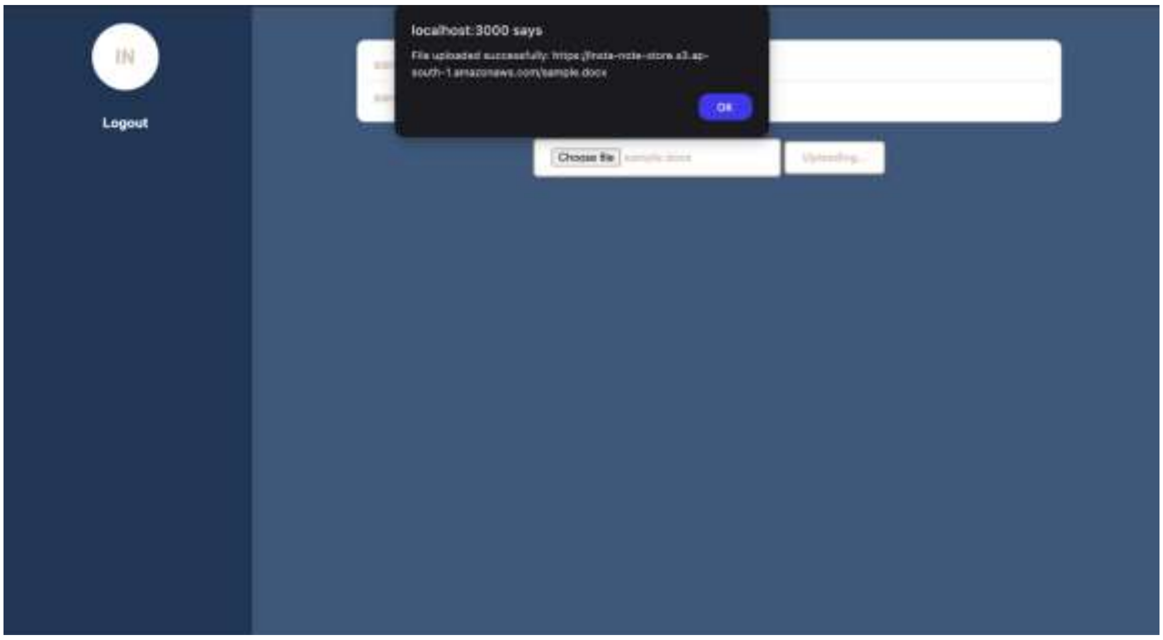


localhost:3000 says

File uploaded successfully: https://insta-note-store.s3.ap-south-1.amazonaws.com/sample.zip

Result: Successfully uploaded a ZIP file.

Test 3: Uploading a DOCX file.


localhost:3000 says

File uploaded successfully: https://insta-note-store.s3.ap-south-1.amazonaws.com/sample.docx

Result: Successfully uploaded a DOCX file.

7. SCOPE FOR FUTURE ENCHANCEMENT

Scope for Future Enhancement

- **Enhanced Security:** Implement advanced security measures such as two-factor authentication, encryption of files at rest and in transit, and user activity logging for better monitoring and auditing.

- **User Roles and Permissions:** Develop a role-based access control system to differentiate user permissions, allowing for more granular control over who can upload, download, and manage files.

- **File Versioning:** Introduce a file versioning system to keep track of changes to files and allow users to revert to previous versions if needed.

- **Collaboration Features:** Add features that support collaboration, such as file sharing with specific users, real-time editing, and commenting on files.

- **Mobile Support:** Create a mobile-friendly version of the platform or develop a dedicated mobile app to make it accessible on various devices.

- **Analytics and Reporting:** Provide users with insights and analytics on their file usage, including storage statistics, file access logs, and activity reports.

8. CONCLUSION

In conclusion, the "InstaNote" project showcases a robust and scalable solution for managing and sharing files using AWS technologies. By leveraging services such as Amazon S3 for storage, Node.js for backend processing, and front-end technologies like HTML, CSS, and JavaScript, this platform delivers a user-friendly experience for file management.
The modular design ensures easy maintenance and the potential for future enhancements. This project not only demonstrates the practical application of cloud computing and web development but also lays a strong foundation for further development and innovation in the domain of file sharing and collaboration.

9. BIBILOGRAPHY

Books referred: AWS Cloud Projects: Strengthen your AWS skills through practical projects, from websites to advanced AI applications by **Ivo Pinto and Pedro Santos**

Articles:

Sources: https://bluexp.netapp.com/blog/aws-blg-4-aws-file-storage-solutions-and-how-to-make-the-most-of-them

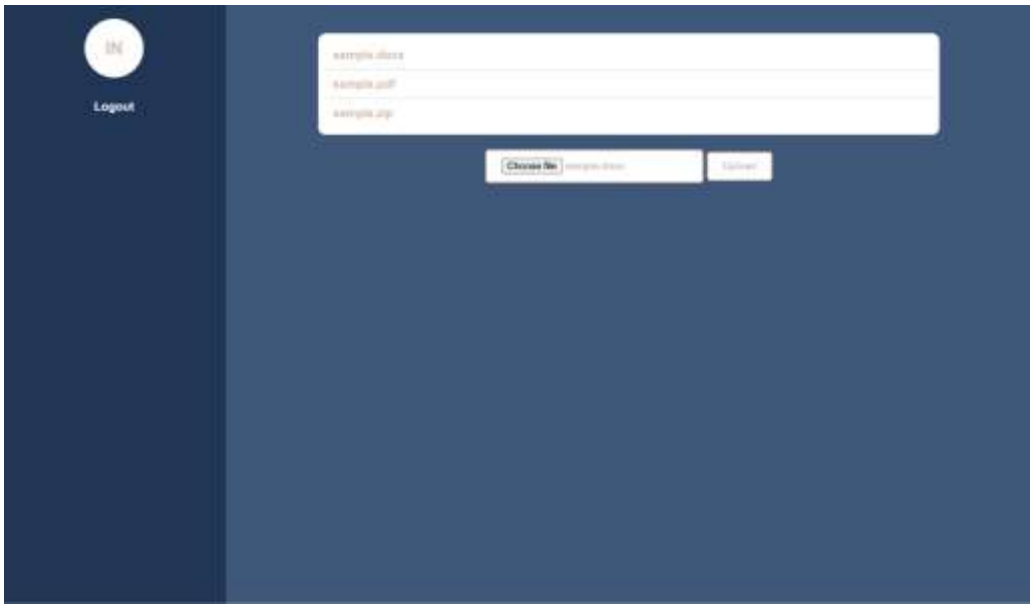
YouTube tutorials:

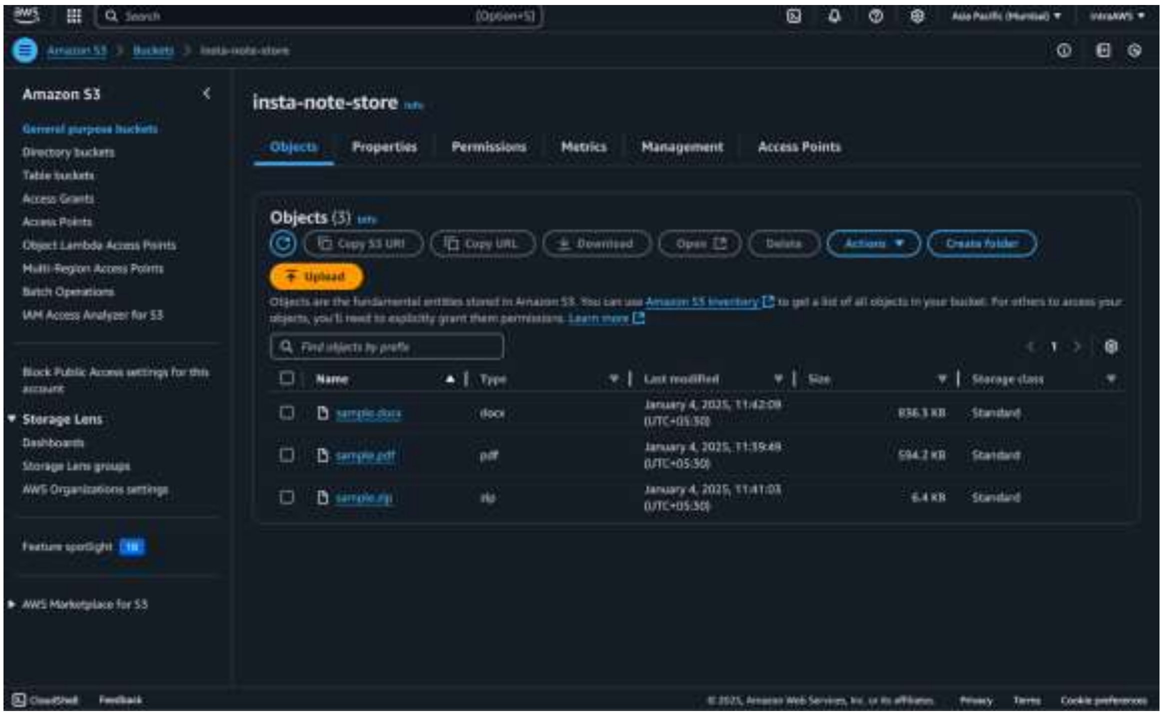
Source: https://www.youtube.com/watch?v=sCQwEVhCvTg

Web Reference:

Source: https://aws.amazon.com/what-is/cloud-file-storage/

10. SNAPSHOTS



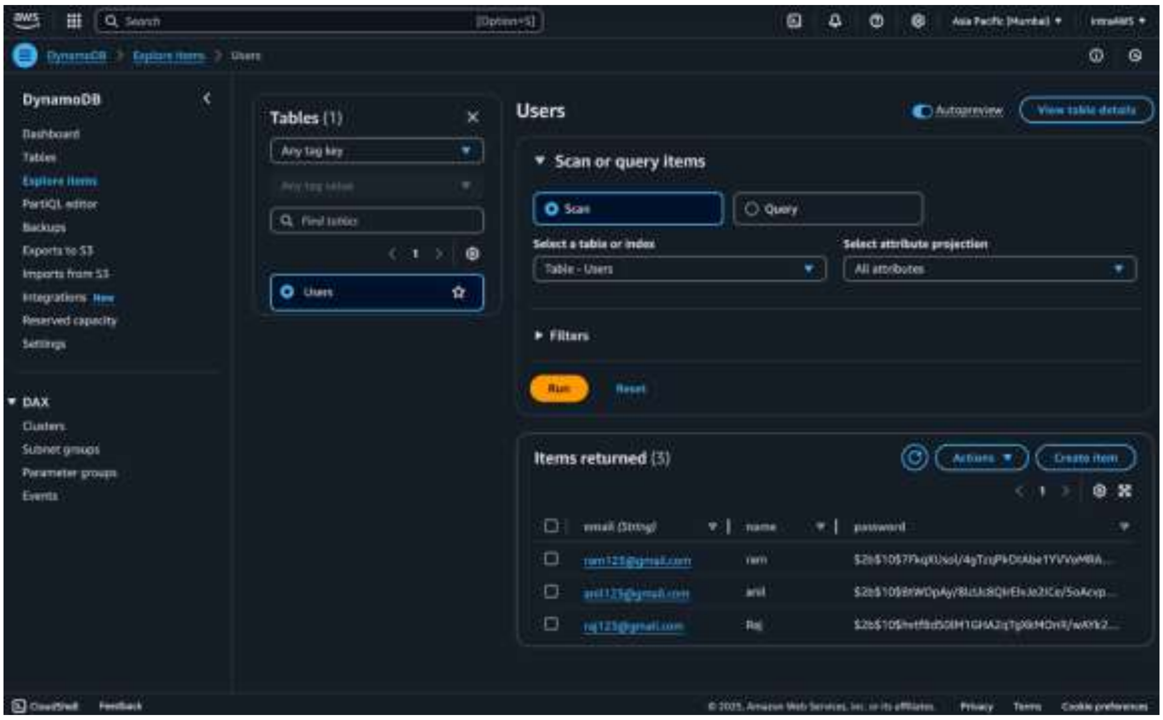


insta-note-store

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
sample.docx	docx	January 4, 2025, 11:42:08 (UTC+05:30)	836.3 KB	Standard
sample.pdf	pdf	January 4, 2025, 11:39:49 (UTC+05:30)	594.2 KB	Standard
sample.zip	zip	January 4, 2025, 11:41:03 (UTC+05:30)	6.4 KB	Standard



Users

Scan or query items

Scan

Select a table or index: Table - Users

Select attribute projection: All attributes

Items returned (3)

email (String)	name	password
ram123@gmail.com	ram	$2b$10$7FkqKUsol/4gTzqPkDtAbe1YVVoMBA...
anil123@gmail.com	anil	$2b$10$BtWDpAy/8lzUc8QirElxJe2ICe/5oAcvp...
raj123@gmail.com	Raj	$2b$10$hetf8d5OlH1GHA2qTg9kMOnX/wAYk2...