

Lab Activity (Serverless)

### Submitted by: Merin Mary Saju B00943351

Contents

[AIM 2](#_bookmark0)

[Thought Process 3](#_bookmark1)

[Challenges Faced: 3](#_bookmark2)

[Design Considerations 3](#_bookmark3)

[Deployed URL 4](#_bookmark4)

[Output Screenshots 4](#_bookmark5)

[References 7](#_bookmark6)

# AIM

I have successfully developed a serverless application that meets all the specified objectives, utilizing four Lambda functions, two DynamoDB use cases, and one S3 use case. For this project, I created an Employee Management Website that serves as a dashboard for creating employees, displaying all employee details, and deleting employees and storing their images in S3 and other personal information in DynamoDB.

This activity has been instrumental in enhancing my skills in building serverless applications using AWS services such as S3, Lambda, and DynamoDB. Additionally, it provided valuable experience in integrating these AWS services with a React application.

# Thought Process

I have successfully developed a serverless application that meets all specified objectives, utilizing four Lambda functions, two DynamoDB use cases, and one S3 use case. The project is an Employee Management Website that functions as a dashboard for creating employees, displaying all employee details, and deleting employees. The frontend UI is implemented using the React framework, with Axios used for making calls to backend services. The UI consists of a basic form to collect employee details such as name, employee ID, address, department, experience, and an option to upload their image.

Once the data is collected, the application first calls the s3upload Lambda function to store the image in S3, which returns a publicly accessible S3 URL [1]. With the S3 URL obtained, the application sends all other information, along with the URL, to API Gateway, which invokes the addemployee Lambda function. This function processes the JSON data and stores the details in DynamoDB using the AWS SDK [2]. The UI populates all employee details using the getallemployee Lambda function, called via another API Gateway. There is also an option in the UI to delete employee records, handled by the deleteemployee Lambda function, which interacts with DynamoDB via API Gateway [2].

## Challenges Faced:

I faced mainly two challenges in developing the application:

* I was getting CORS issue when calling the AWS services from React UI. This was resolved with help of returning necessary CORS headers back to the React-UI
* Also faced issues with s3 upload, getting access denied after uploading. This was resolved by adding bucket policies with necessary read policies for everyone.

## Design Considerations

The design considerations for this application include several key aspects. Scalability is achieved through the use of AWS Lambda, which ensures that the application can scale automatically with the number of requests without the need to manage servers [1].

DynamoDB provides scalable storage with high availability and durability [2]. Additionally, API Gateway and Lambda functions use IAM roles and policies to restrict access to the

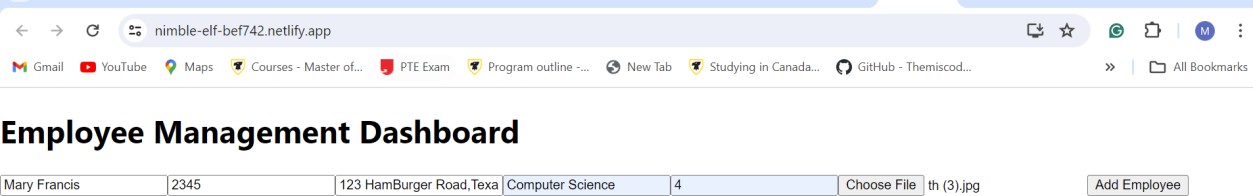
resources. Performance is optimized by using AWS Lambda and API Gateway, which ensures low latency and high performance for API calls, and DynamoDB's fast and predictable performance further enhances the user experience. Cost efficiency is achieved through the serverless architecture, leveraging AWS Lambda’s pay-as-you-go model, ensuring there are no charges for idle resources and costs scale with usage. Maintainability is enhanced by designing the application with modular Lambda functions, each handling specific tasks such as uploading to S3, adding employees to DynamoDB, fetching employee details, and deleting employees, which simplifies maintenance and updates. Finally, the user experience is

prioritized with a React-based frontend providing a responsive and dynamic user interface, and Axios ensuring smooth and efficient communication between the frontend and backend services.

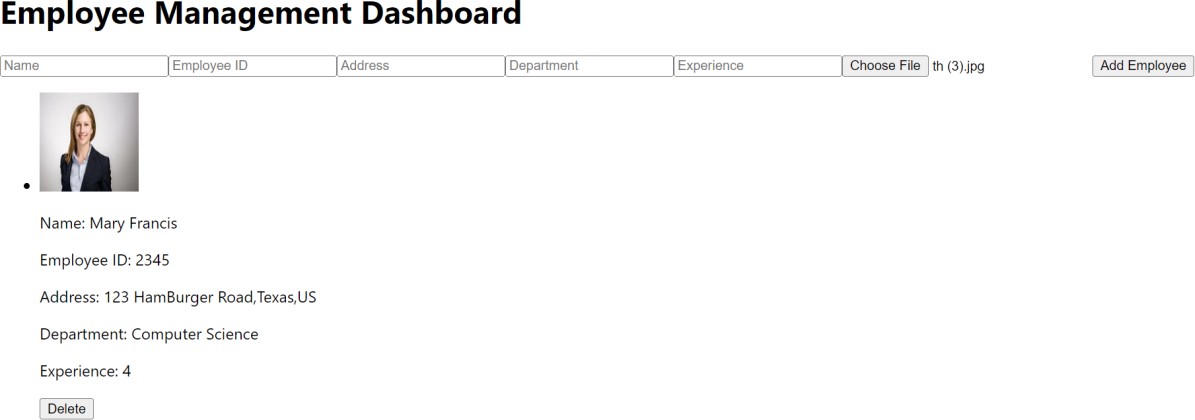
# Deployed URL

#### <https://nimble-elf-bef742.netlify.app/>

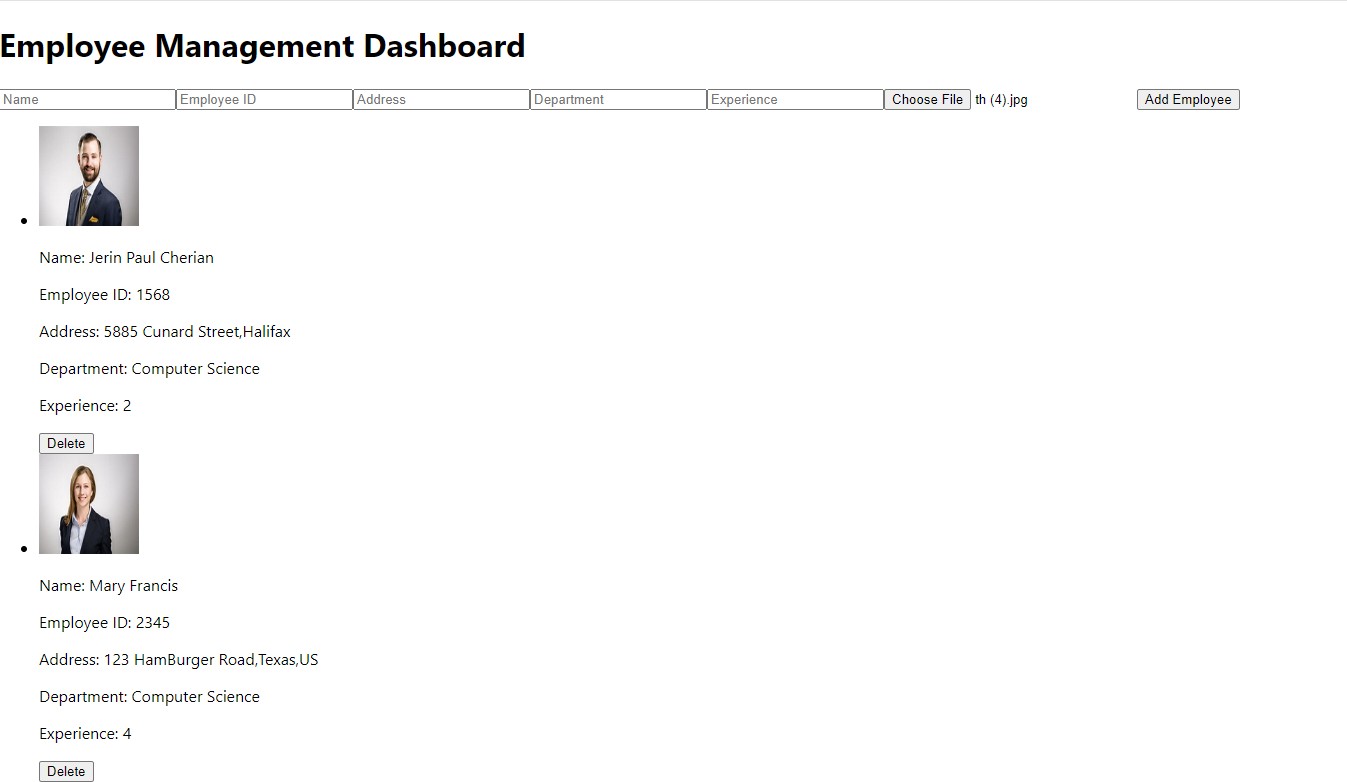
# Output Screenshots



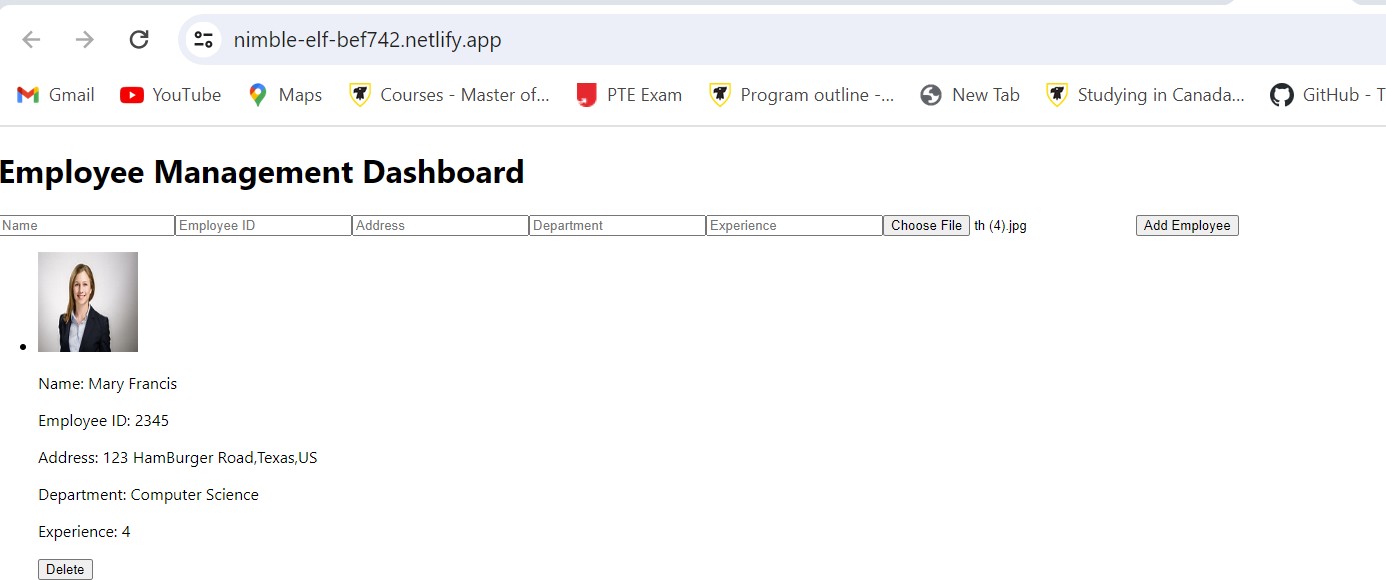
*Fig 1 shows the output screenshot of UI after entering the form details*



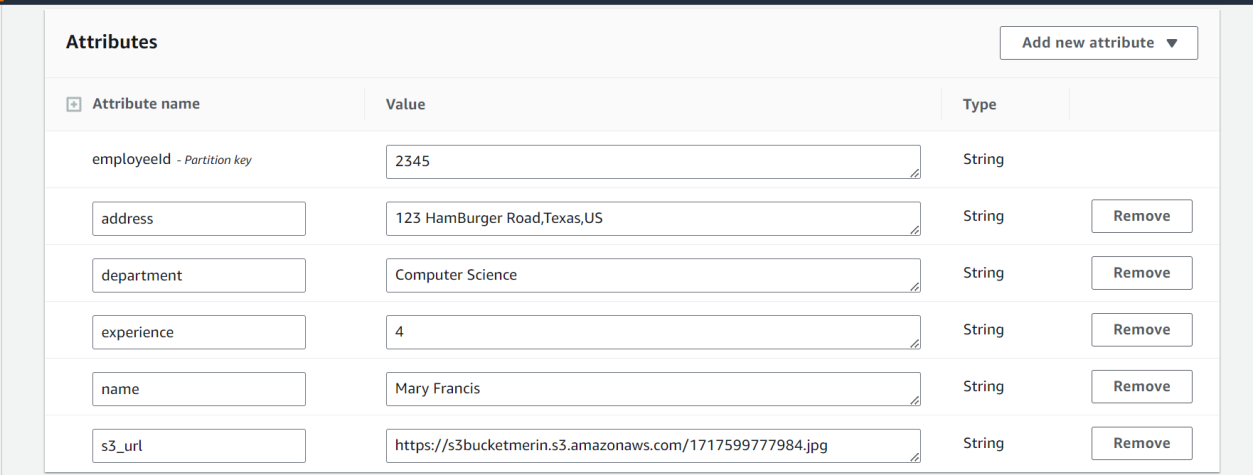
*Fig 2 shows the output screenshot of UI after clicking the add employee button*



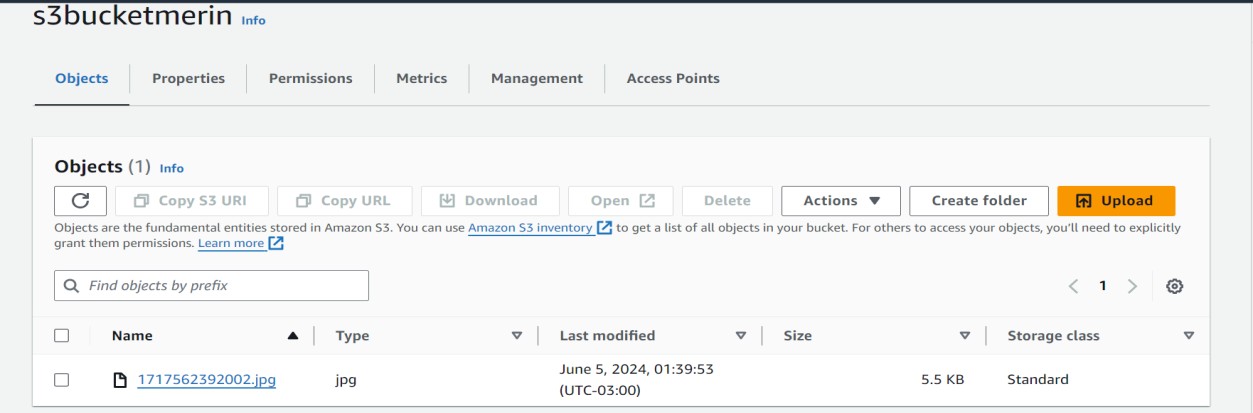
*Fig 3 shows the output screenshot of UI after clicking the another add employee button*



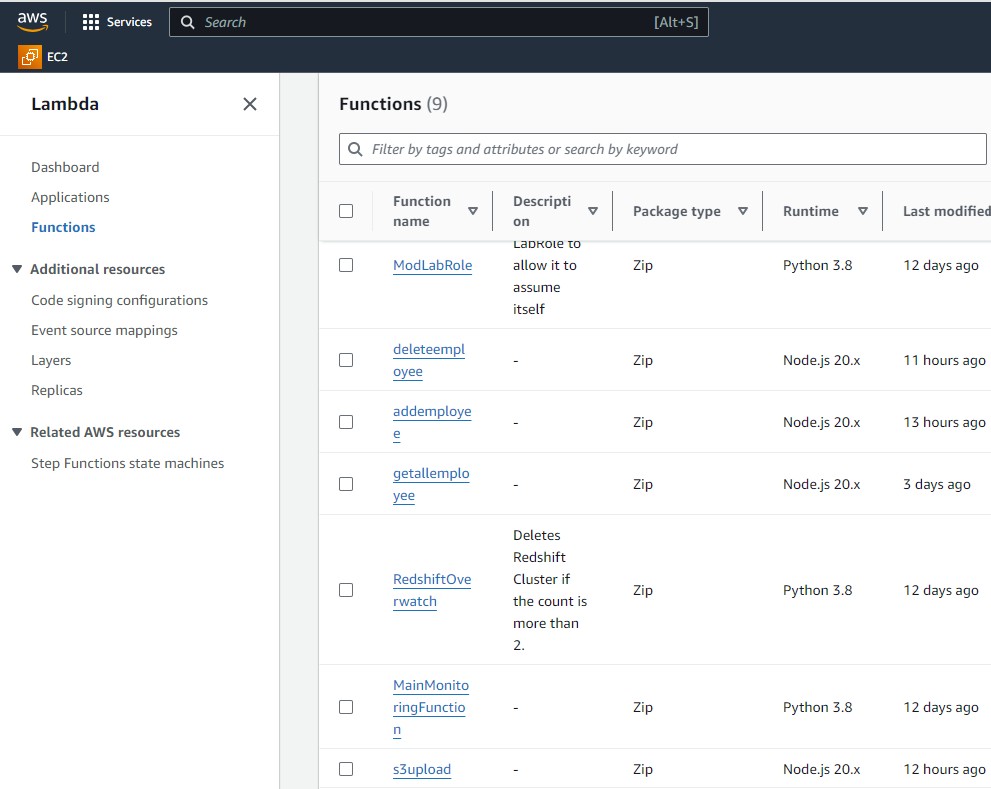
*Fig 4 shows the output screenshot of UI after clicking the delete button*



*Fig 5 shows the output screenshot of DynamoDB*



*Fig 6 shows the output screenshot of S3*



*Fig 7 shows the output screenshot of Lambda functions*

# References

1. R. MacCárthaigh, "Building scalable serverless applications with Amazon S3 and AWS Lambda," AWS Compute Blog, Jun. 05, 2024. [Online]. Available:

[https://aws.amazon.com/blogs/compute/building-scalable-serverless-applications-with-amazon-s3-](https://aws.amazon.com/blogs/compute/building-scalable-serverless-applications-with-amazon-s3-and-aws-lambda/) [and-aws-lambda/.](https://aws.amazon.com/blogs/compute/building-scalable-serverless-applications-with-amazon-s3-and-aws-lambda/) Accessed: Jun. 03, 2024.

1. "Using AWS Lambda with Amazon DynamoDB," AWS Lambda Developer Guide, Jun. 05, 2024. [Online]. Available: [https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html.](https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html) Accessed: Jun. 02, 2024.