DALHOUSIE UNIVERSITY

CSCI 5409

Advance Topics in Cloud Computing

Md Samshad Rahman

B00968344

Deployed URLs:

http://ec2-52-90-42-45.compute-1.amazonaws.com/

Project Description:

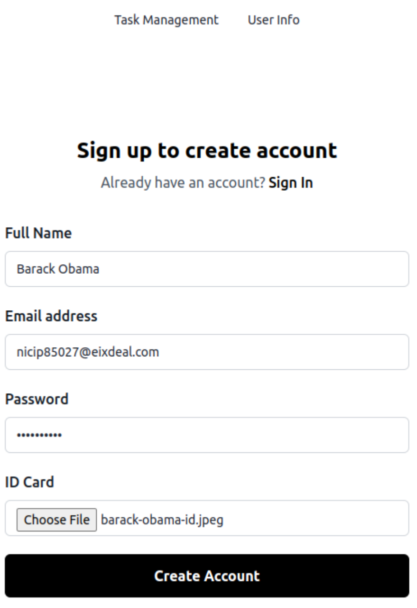
SecureTask: Identity Verification and Task Management System

SecureTask is a cutting-edge web application designed to seamlessly integrate identity verification with robust task management. The platform empowers users to securely register, verify their identities, and efficiently manage their tasks through an intuitive interface.

Features

1. **Identity Verification System**
    - **User Registration**: Users sign up by providing their name, email, and an ID card photo.
    - **Selfie Verification**: Users verify their accounts by uploading a selfie. The system uses AWS Rekognition to match the selfie with the ID card photo, ensuring an accuracy rate of 80% or higher for successful verification.
2. **Task Management System**
    - **Create Tasks**: Users can create tasks with detailed descriptions and deadlines.
    - **Manage Tasks**: Organize tasks by priority, due date, and status.
    - **Track Progress**: Monitor the completion status of tasks and update progress in real-time.

Some screenshots of the application:



Sign up to create account

Already have an account? **Sign In**

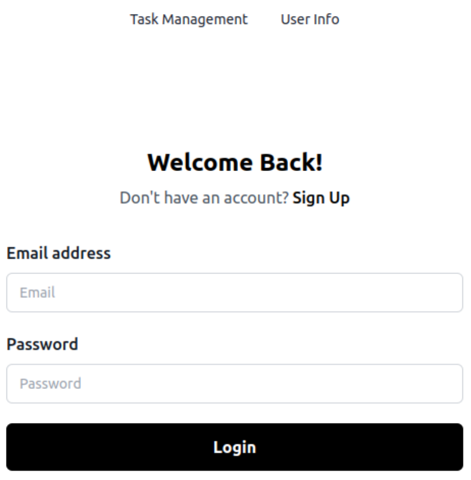
Full Name: Barack Obama

Email address: nicip85027@eixdeal.com

Password: ••••••••••

ID Card: barack-obama-id.jpeg

Figure 1 User Registration



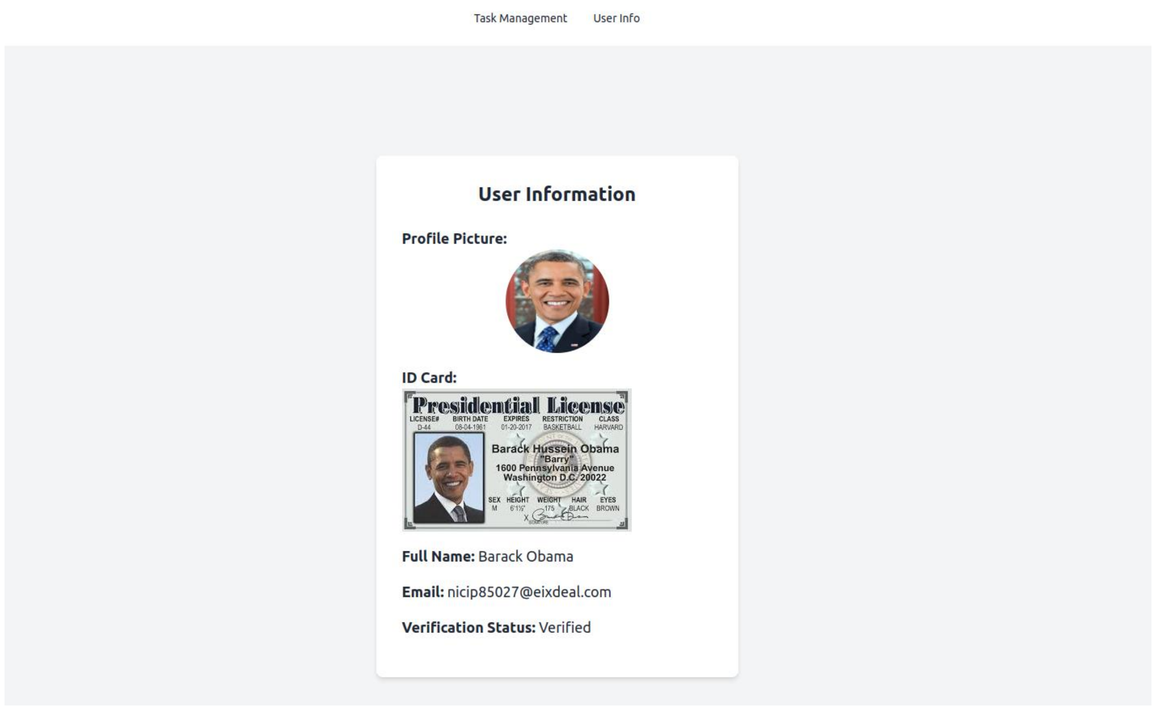
Welcome Back!

Don't have an account? **Sign Up**

Email address: Email

Password: Password

Figure 2 User Login



User Information

Profile Picture:

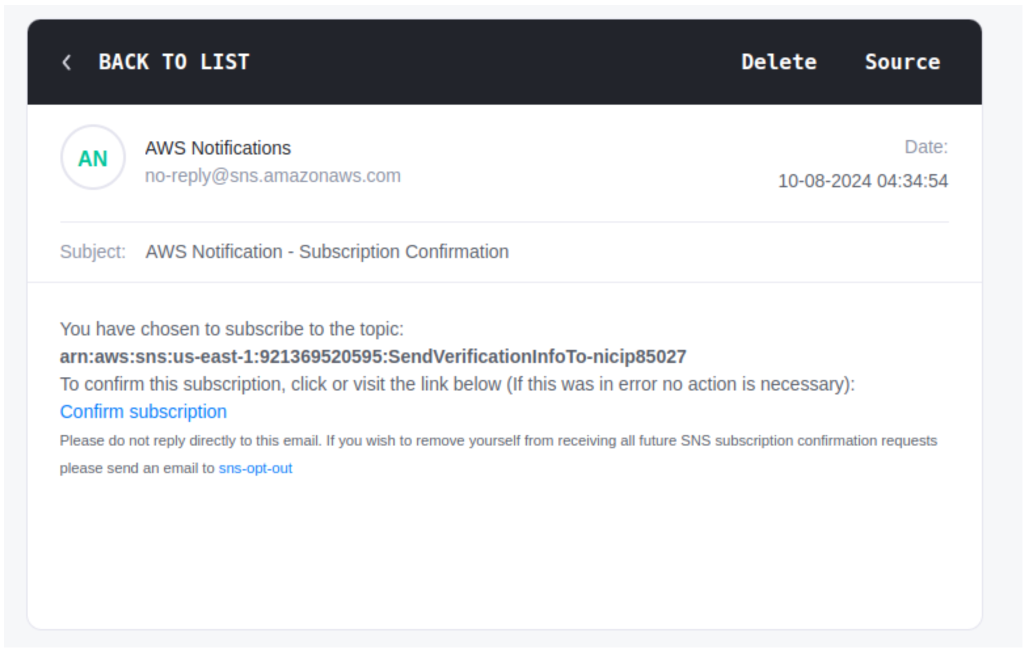
ID Card:

Full Name: Barack Obama

Email: nicip85027@eixdeal.com

Verification Status: Verified

Figure 3 User Information (After Verification)



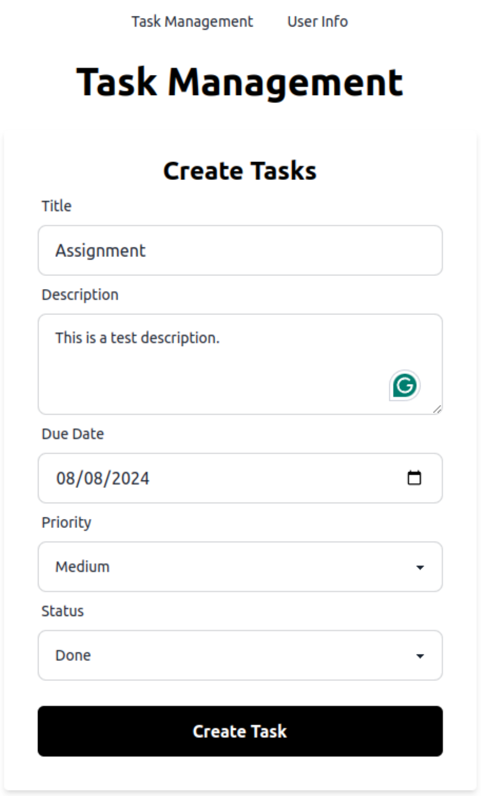
AWS Notifications
no-reply@sns.amazonaws.com

Date:
10-08-2024 04:34:54

Subject: AWS Notification - Subscription Confirmation

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:921369520595:SendVerificationInfoTo-nicip85027
To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription
Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

Figure 4 Verification Notification



Task Management   User Info

Task Management

Create Tasks

Title: Assignment

Description: This is a test description.

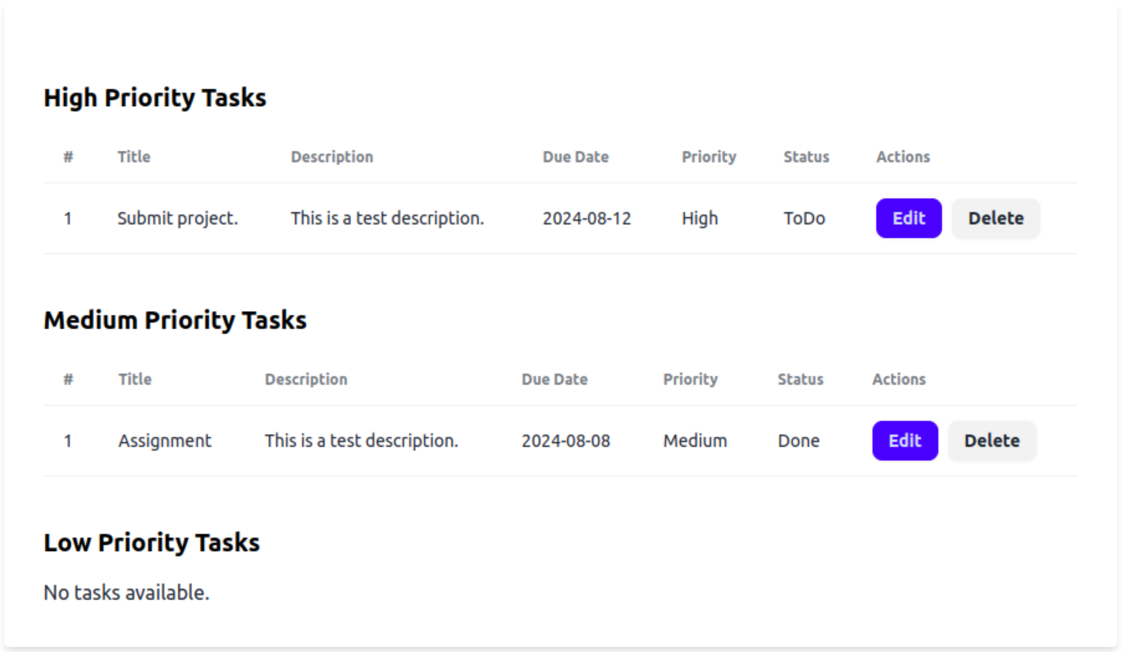
Due Date: 08/08/2024

Priority: Medium

Status: Done

Create Task

Figure 5 Task Create



High Priority Tasks

#	Title	Description	Due Date	Priority	Status	Actions
1	Submit project.	This is a test description.	2024-08-12	High	ToDo	Edit Delete

Medium Priority Tasks

#	Title	Description	Due Date	Priority	Status	Actions
1	Assignment	This is a test description.	2024-08-08	Medium	Done	Edit Delete

Low Priority Tasks

No tasks available.

Figure 6 Task Management Dashboard

Requirements:
I have used the following services to develop the SecureTask web application:

Compute (2):

1. **AWS Lambda:**
   AWS Lambda was chosen for handling registration, image processing, and verification due to its serverless architecture. Lambda enables me to run code without having to provision or manage servers, making it ideal for executing backend processes on-demand. It also scales automatically depending on the volume of requests, ensuring that resources are used efficiently and cost effectively. This service works nicely with other AWS services like API Gateway and DynamoDB, resulting in a unified environment for my application.

   Alternatives Considered:
   - AWS Fargate: Runs containers without managing servers, but Lambda's event-driven model is more efficient for my needs.
   - Amazon EC2: Requires manual scaling and management, making Lambda a more efficient choice for serverless computing tasks.

2. **AWS EC2:**
   AWS EC2 was chosen to host frontend services because it provides resizable computational capacity in the cloud, allowing us the flexibility and control required to host and operate these services effectively. EC2's ability to manage persistent compute resources makes it an excellent fit for my application.

   Alternatives Considered:
   - AWS Fargate: More suitable for containerized applications, whereas EC2 provides the flexibility and control needed for hosting frontend services.

Storage (2):

1. **Amazon S3:**
   Amazon S3 was chosen for storing uploaded selfies and ID card photos in jpg/jpeg format. because of its scalable storage options that are both durable and reliable. S3 is ideal for safely and efficiently storing user-uploaded photos. Its smooth connection with other AWS services, including Lambda and Rekognition, increases its usefulness in my project.

   Alternatives Considered:
   - EBS (Elastic Block Store): Provides block storage, which is less suitable for scalable object storage needs.
   - Amazon Glacier: Designed for long-term archival storage, not for frequently accessed image files.

2. **Amazon DynamoDB:**
   DynamoDB was chosen to store user data and verification status because of its rapid and consistent performance, as well as its smooth scalability. It is appropriate for storing verification data due to its low latency and high availability, which ensure that the application remains responsive and reliable.

   Alternatives Considered:
   - Amazon RDS: Suitable for relational databases, whereas DynamoDB's NoSQL architecture is more appropriate for the application's requirements.
   - Amazon Aurora: Also, a relational database, more suited for applications needing complex transactions and query capabilities.

Network (1):

1. **AWS API Gateway:**
   AWS API Gateway was selected because of its ability to simplify API creation and management. It interacts smoothly with AWS Lambda, making it easier to create serverless APIs that handle user requests efficiently. API Gateway eliminates the need for manual infrastructure configuration and server maintenance, speeding the development process.

General (3):

1. **AWS CloudFormation:**
   AWS CloudFormation was selected for its ability to automate and simplify the management of AWS infrastructure through reusable templates. Its features ensure consistency, reduce manual effort, and provide robust management of dependencies and rollbacks. This makes CloudFormation an ideal choice for maintaining scalable, reliable, and repeatable infrastructure configurations in the application.

2. **AWS Rekognition:**
   Amazon Rekognition was selected for comparing selfies with ID card photos because of its advanced image analysis capabilities. It ensures high accuracy and reliability in matching images, which is crucial for the identity verification process in my application.

   Alternatives Considered:
   - Amazon SageMaker: While SageMaker offers comprehensive machine learning capabilities, including custom image analysis models, it requires more development effort and expertise to build and maintain custom models for facial recognition. Rekognition, on the other hand, provides a ready-to-use, fully managed solution that is specifically optimized for image and video analysis, making it a more efficient choice for this particular use case.

3. Amazon SNS:

Amazon SNS was chosen for sending notifications to users regarding the status of their verification and task updates due to its reliable and timely message delivery capabilities.

Alternatives Considered:

- Amazon SQS: A message queuing service better suited for decoupling microservices, not for direct user notifications.

Deployment Model:

I have chosen the public cloud for my identity verification and task management system due to its accessibility, scalability, and security. This model ensures that users worldwide can seamlessly register, verify identities, and manage tasks. AWS handles automatic scaling and infrastructure management, which maintains performance and reliability during peak times. Robust security measures, including encryption and firewalls, protect sensitive data like ID card photos and selfies. Additionally, the public cloud is cost-efficient and allows for easy replication across regions and access to new services. This choice simplifies infrastructure management and enhances the application's functionality and user experience.

Delivery Model:

My application operates under the Software as a Service (SaaS) model. I've used in combination of AWS Lambda (Function as a Service), DynamoDB (Database as a Service), S3 (Storage as a Service), and EC2 (Infrastructure as a Service) to build and deliver the service. API Gateway exposes the necessary endpoints, which trigger Lambda functions to handle computations, interact with other services, and deliver responses. This setup ensures that there's no manual infrastructure management—my code runs only when needed, and there are no idle resources. The system is efficient and scalable, providing on-demand functionality without the overhead of managing physical servers.

Architecture Overview:

The architecture of SecureTask seamlessly integrates various AWS services to enable secure identity verification and efficient task management. Users interact with the platform through a React application hosted on an EC2 instance, accessing it via their devices. This front-end application allows users to register, upload selfies and ID card photos, and manage tasks. It communicates with the backend through the API Gateway, which routes requests to AWS Lambda functions. All computations are performed within Lambda functions. These serverless functions handle the core backend operations, such as user registration, image processing, and verification, by interacting with Amazon DynamoDB, S3, and Rekognition. Amazon Rekognition is utilized to analyze and match selfies with ID card photos for verification, while lambda Layer zip files, Images are stored in an S3 bucket. User data is maintained in DynamoDB.

To provision the entire architecture, I utilized CloudFormation, using YAML for its cleaner syntax compared to JSON. All computations within the Lambda functions require code, and I chose Python for this purpose because it offers extensive libraries, including boto3, which simplifies communication with other AWS services.

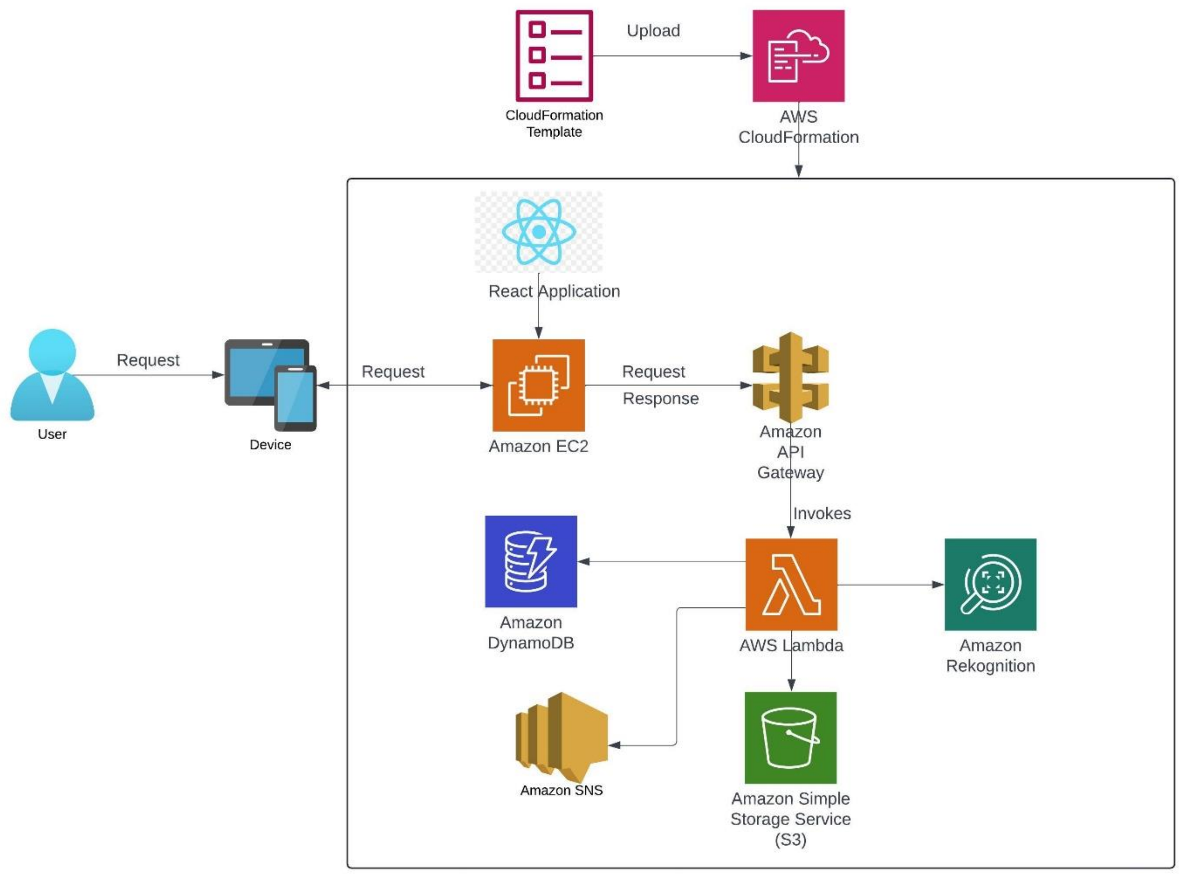


Figure 7 SecureTask application's System Architecture

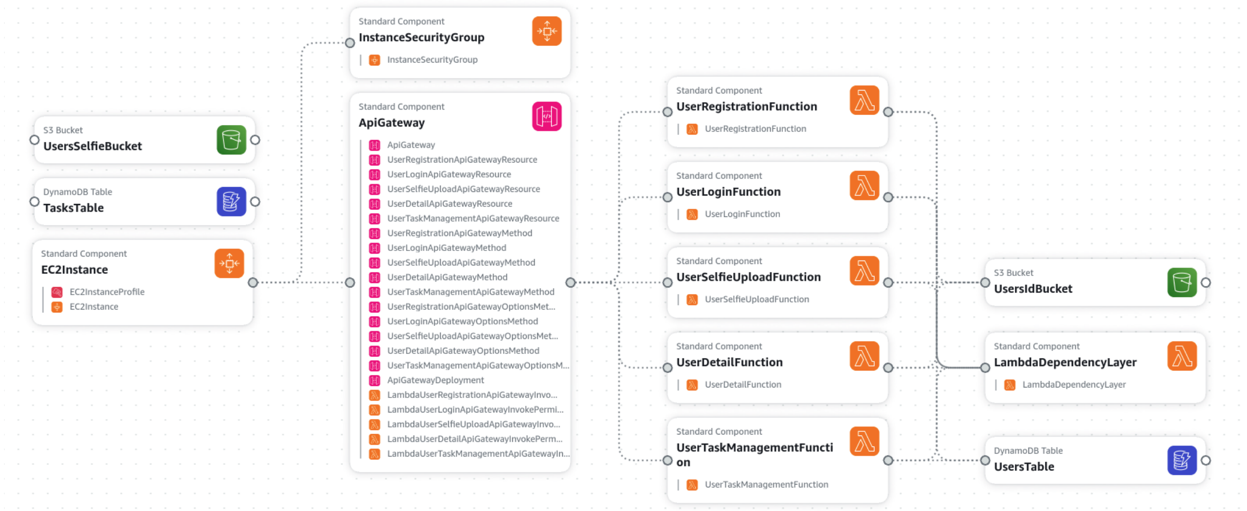

S3 Bucket
UsersSelfieBucket
DynamoDB Table
TasksTable
Standard Component
EC2Instance
EC2InstanceProfile
EC2Instance
Standard Component
InstanceSecurityGroup
InstanceSecurityGroup
Standard Component
ApiGateway
ApiGateway
UserRegistrationApiGatewayResource
UserLoginApiGatewayResource
UserSelfieUploadApiGatewayResource
UserDetailApiGatewayResource
UserTaskManagementApiGatewayResource
UserRegistrationApiGatewayMethod
UserLoginApiGatewayMethod
UserSelfieUploadApiGatewayMethod
UserDetailApiGatewayMethod
UserTaskManagementApiGatewayMethod
UserRegistrationApiGatewayOptionsMet...
UserLoginApiGatewayOptionsMethod
UserSelfieUploadApiGatewayOptionsMet...
UserDetailApiGatewayOptionsMethod
UserTaskManagementApiGatewayOptionsM...
ApiGatewayDeployment
LambdaUserRegistrationApiGatewayInvo...
LambdaUserLoginApiGatewayInvokePermi...
LambdaUserSelfieUploadApiGatewayInvo...
LambdaUserDetailApiGatewayInvokePerm...
LambdaUserTaskManagementApiGatewayIn...
Standard Component
UserRegistrationFunction
UserRegistrationFunction
Standard Component
UserLoginFunction
UserLoginFunction
Standard Component
UserSelfieUploadFunction
UserSelfieUploadFunction
Standard Component
UserDetailFunction
UserDetailFunction
Standard Component
UserTaskManagementFuncti on
UserTaskManagementFunction
S3 Bucket
UsersIdBucket
Standard Component
LambdaDependencyLayer
LambdaDependencyLayer
DynamoDB Table
UsersTable

Figure 8 Designer View in Cloudformation

Security:

SecureTask relies heavily on AWS services for security. User passwords are stored encrypted in Amazon DynamoDB, ensuring full encryption of data at rest. To protect data in transit, all AWS services used support TLS protocol encryption. Security groups act as virtual firewalls for the EC2 instance hosting the web application's frontend. The backend, running on AWS Lambda, is secured by AWS API Gateway with JWT authentication implemented for REST APIs. While a VPC could provide additional security, I've made the APIs public for now to allow testing through Postman and to ensure the application can run on both mobile devices and computers.

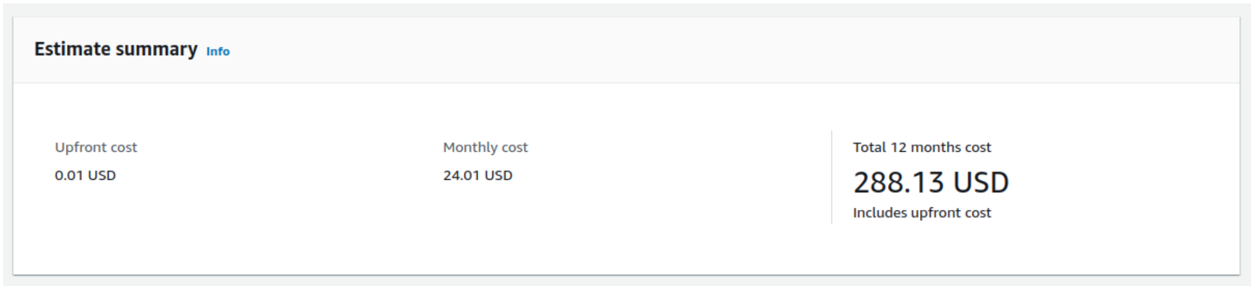
Cost metrics:

I've used the AWS Pricing Calculator to calculate estimated costs for 500 users.

Below are the cost indicators for my project:

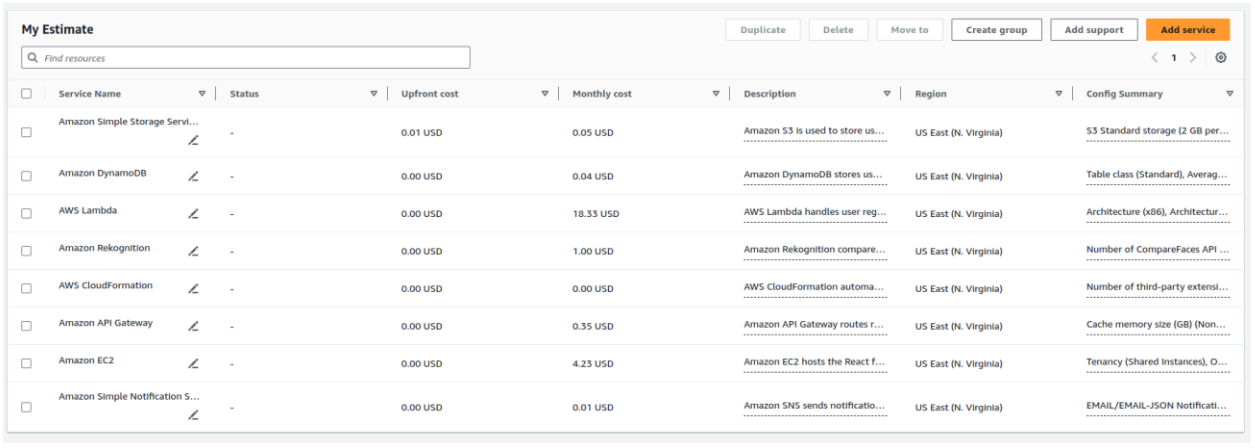
Current Estimates:

Cost metrics URL:
https://calculator.aws/#/estimate?id=f5a117795d826067ca80f5aaa485d1bf7913c2a2



Upfront cost	Monthly cost	Total 12 months cost
0.01 USD	24.01 USD	288.13 USD
		Includes upfront cost

Figure 9 Upfront, monthly and annual cost (approx.)



Service Name	Status	Upfront cost	Monthly cost	Description	Region	Config Summary
Amazon Simple Storage Servi...	-	0.01 USD	0.05 USD	Amazon S3 is used to store us...	US East (N. Virginia)	S3 Standard storage (2 GB per...
Amazon DynamoDB	-	0.00 USD	0.04 USD	Amazon DynamoDB stores us...	US East (N. Virginia)	Table class (Standard), Averag...
AWS Lambda	-	0.00 USD	18.33 USD	AWS Lambda handles user reg...	US East (N. Virginia)	Architecture (x86), Architectur...
Amazon Rekognition	-	0.00 USD	1.00 USD	Amazon Rekognition compare...	US East (N. Virginia)	Number of CompareFaces API ...
AWS CloudFormation	-	0.00 USD	0.00 USD	AWS CloudFormation automa...	US East (N. Virginia)	Number of third-party extensi...
Amazon API Gateway	-	0.00 USD	0.35 USD	Amazon API Gateway routes r...	US East (N. Virginia)	Cache memory size (GB) (Non...
Amazon EC2	-	0.00 USD	4.23 USD	Amazon EC2 hosts the React f...	US East (N. Virginia)	Tenancy (Shared Instances), O...
Amazon Simple Notification S...	-	0.00 USD	0.01 USD	Amazon SNS sends notificatio...	US East (N. Virginia)	EMAIL/EMAIL-JSON Notificati...

Figure 10 Detailed view for the current estimate

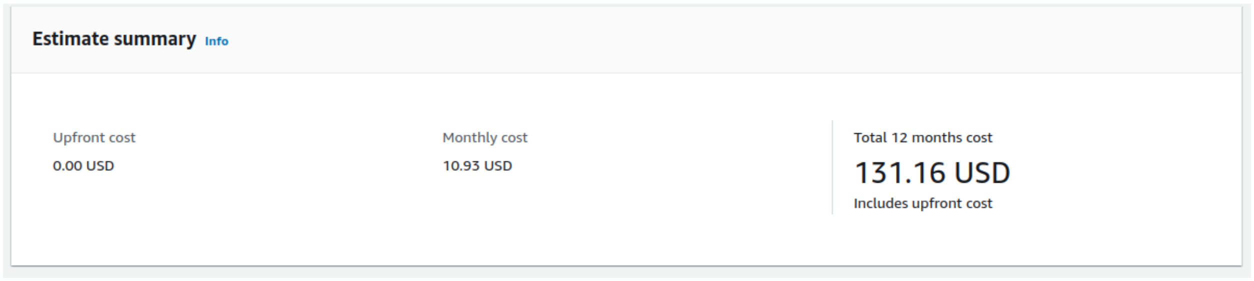
From above figures 9 and 10, there will be 0.01 USD upfront cost, and the monthly cost will be 24.01 USD.

Cheaper Alternative Estimates:

To further reduce the cost of running SecureTask, I can implement several cost-saving strategies:

- Optimizing Lambda Memory Allocation: Fine-tuning the memory allocated to AWS Lambda functions.
- Reducing Request Duration: Minimizing the execution time of each request.
- Choosing EC2 Instance Savings Plans: Selecting EC2 Instance Savings Plans instead of Compute Savings Plans.

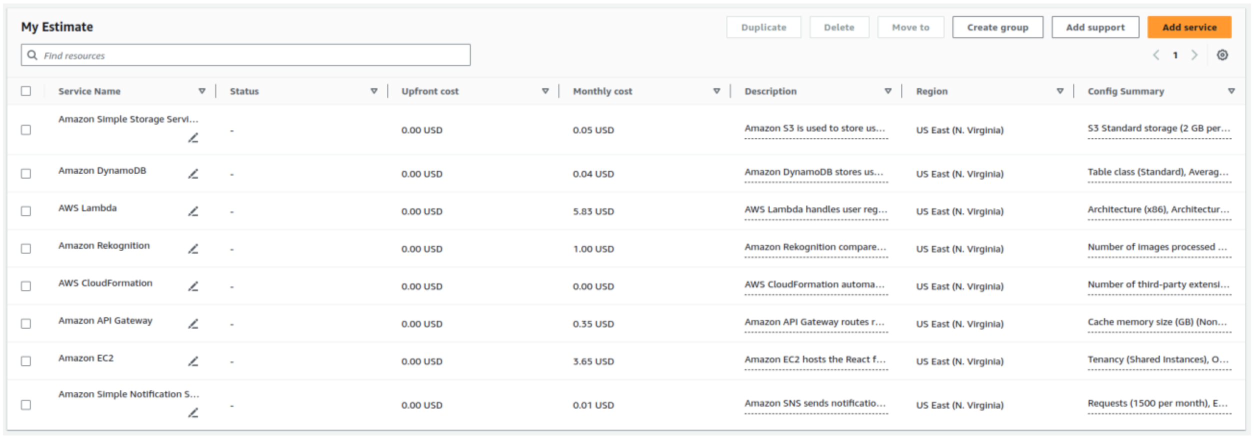
Cost metrics URL:
https://calculator.aws/#/estimate?id=9af7adc7f14a4bd61b029a8e425a9157973da3cb



Estimate summary Info

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	10.93 USD	131.16 USD Includes upfront cost

Figure 121 Upfront, monthly and annual cost (approx.) for cheaper approach



Service Name	Status	Upfront cost	Monthly cost	Description	Region	Config Summary
Amazon Simple Storage Servi...	-	0.00 USD	0.05 USD	Amazon S3 is used to store us...	US East (N. Virginia)	S3 Standard storage (2 GB per...
Amazon DynamoDB	-	0.00 USD	0.04 USD	Amazon DynamoDB stores us...	US East (N. Virginia)	Table class (Standard), Averag...
AWS Lambda	-	0.00 USD	5.83 USD	AWS Lambda handles user reg...	US East (N. Virginia)	Architecture (x86), Architectur...
Amazon Rekognition	-	0.00 USD	1.00 USD	Amazon Rekognition compare...	US East (N. Virginia)	Number of images processed ...
AWS CloudFormation	-	0.00 USD	0.00 USD	AWS CloudFormation automa...	US East (N. Virginia)	Number of third-party extensi...
Amazon API Gateway	-	0.00 USD	0.35 USD	Amazon API Gateway routes r...	US East (N. Virginia)	Cache memory size (GB) (Non...
Amazon EC2	-	0.00 USD	3.65 USD	Amazon EC2 hosts the React f...	US East (N. Virginia)	Tenancy (Shared Instances), O...
Amazon Simple Notification S...	-	0.00 USD	0.01 USD	Amazon SNS sends notificatio...	US East (N. Virginia)	Requests (1500 per month), E...

Figure 112 Detailed view for the cheaper approach

By applying these optimizations, the costs can be significantly reduced:

Yearly Cost Reduction:

- Original Yearly Cost: $24.01 * 12 months = $288.13
- Optimized Yearly Cost: $10.93 * 12 months = $131.16
- This results in a yearly cost reduction from $288.13 to $131.16.

Monthly Cost Reduction:

- Original Monthly Cost: $24.01
- Optimized Monthly Cost: $10.93

These measures will help to lower the operational costs of SecureTask effectively.

References:

[1] "*AWS Lambda*", AWS, (Available). https://aws.amazon.com/lambda/ [Last accessed on August 10, 2024].

[2] "*Amazon S3*", AWS, (Available). https://aws.amazon.com/s3/ [Last accessed on August 10, 2024].

[3] "*Amazon DynamoDB*", AWS, (Available). https://aws.amazon.com/dynamodb [Last accessed on August 10, 2024].

[4] "*Amazon API Gateway*", AWS, (Available). https://aws.amazon.com/api-gateway/ [Last accessed on August 10, 2024].

[5] "*AWS CloudFormation*", AWS, (Available). https://aws.amazon.com/cloudformation/ [Last accessed on August 10, 2024].

[6] "*Amazon Rekognition*", AWS, (Available). https://aws.amazon.com/rekognition [Last accessed on August 10, 2024].

[7] "*AWS Pricing Calculator*", AWS, (Available). https://calculator.aws/#/ [Last accessed on August 10, 2024].