\*\*\*

Code originally included in the Amazon Wild Rydes workshop: <https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/module-3/>

\*\*\*

**LAMBDA FUNCTION**

const randomBytes = require('crypto').randomBytes;

const AWS = require('aws-sdk');

const ddb = new AWS.DynamoDB.DocumentClient();

const fleet = [

{

Name: 'Angel',

Color: 'White',

Gender: 'Female',

},

{

Name: 'Gil',

Color: 'White',

Gender: 'Male',

},

{

Name: 'Rocinante',

Color: 'Yellow',

Gender: 'Female',

},

];

exports.handler = (event, context, callback) => {

if (!event.requestContext.authorizer) {

errorResponse('Authorization not configured', context.awsRequestId, callback);

return;

}

const rideId = toUrlString(randomBytes(16));

console.log('Received event (', rideId, '): ', event);

// Because we're using a Cognito User Pools authorizer, all of the claims

// included in the authentication token are provided in the request context.

// This includes the username as well as other attributes.

const username = event.requestContext.authorizer.claims['cognito:username'];

// The body field of the event in a proxy integration is a raw string.

// In order to extract meaningful values, we need to first parse this string

// into an object. A more robust implementation might inspect the Content-Type

// header first and use a different parsing strategy based on that value.

const requestBody = JSON.parse(event.body);

const pickupLocation = requestBody.PickupLocation;

const unicorn = findUnicorn(pickupLocation);

recordRide(rideId, username, unicorn).then(() => {

// You can use the callback function to provide a return value from your Node.js

// Lambda functions. The first parameter is used for failed invocations. The

// second parameter specifies the result data of the invocation.

// Because this Lambda function is called by an API Gateway proxy integration

// the result object must use the following structure.

callback(null, {

statusCode: 201,

body: JSON.stringify({

RideId: rideId,

Unicorn: unicorn,

Eta: '30 seconds',

Rider: username,

}),

headers: {

'Access-Control-Allow-Origin': '\*',

},

});

}).catch((err) => {

console.error(err);

// If there is an error during processing, catch it and return

// from the Lambda function successfully. Specify a 500 HTTP status

// code and provide an error message in the body. This will provide a

// more meaningful error response to the end client.

errorResponse(err.message, context.awsRequestId, callback)

});

};

// This is where you would implement logic to find the optimal unicorn for

// this ride (possibly invoking another Lambda function as a microservice.)

// For simplicity, we'll just pick a unicorn at random.

function findUnicorn(pickupLocation) {

console.log('Finding unicorn for ', pickupLocation.Latitude, ', ', pickupLocation.Longitude);

return fleet[Math.floor(Math.random() \* fleet.length)];

}

function recordRide(rideId, username, unicorn) {

return ddb.put({

TableName: 'Rides',

Item: {

RideId: rideId,

User: username,

Unicorn: unicorn,

RequestTime: new Date().toISOString(),

},

}).promise();

}

function toUrlString(buffer) {

return buffer.toString('base64')

.replace(/\+/g, '-')

.replace(/\//g, '\_')

.replace(/=/g, '');

}

function errorResponse(errorMessage, awsRequestId, callback) {

callback(null, {

statusCode: 500,

body: JSON.stringify({

Error: errorMessage,

Reference: awsRequestId,

}),

headers: {

'Access-Control-Allow-Origin': '\*',

},

});

}

**TEST EVENT FOR LAMBDA FUNCTION**

{

"path": "/ride",

"httpMethod": "POST",

"headers": {

"Accept": "\*/\*",

"Authorization": "eyJraWQiOiJLTzRVMWZs",

"content-type": "application/json; charset=UTF-8"

},

"queryStringParameters": null,

"pathParameters": null,

"requestContext": {

"authorizer": {

"claims": {

"cognito:username": "the\_username"

}

}

},

"body": "{\"PickupLocation\":{\"Latitude\":47.6174755835663,\"Longitude\":-122.28837066650185}}"

}