MOO:
Backend Technical Assignment

Technical Design Document

For MOO

Client	MOO
Project	MOO Backend Technical Assignment
Countries and languages	Worldwide / English

Contents

Technical Overview

The task is to create an API that our clients can use to access information about their customers' orders.

They should be able to queries things like price and name for the product, the items which are included in an order and all the given orders for one client.

Given MOO's tech stack this will be solved using AWS functionality.

Technologies Used

The API will be solved using what are for MOO standard technologies:

- AWS Cloud
  - Amazon DynamoDB
  - Amazon API Gateway
  - Lambda functions written in TypeScript and compiled into JavaScript

All are licensed to use with Amazon WebServices

Back-end technologies

Technology	License	Notes
Amazon API Gateway		Comes with OIDC, Oauth2 and native CORS support
Amazon Lambda Functions		Used to add functionality to the exposed routes. Written in TypeScript compiled to JavaScript
Amazon DynamoDB		Data storage

Front-end technologies

N/A as this is pure backend functionality

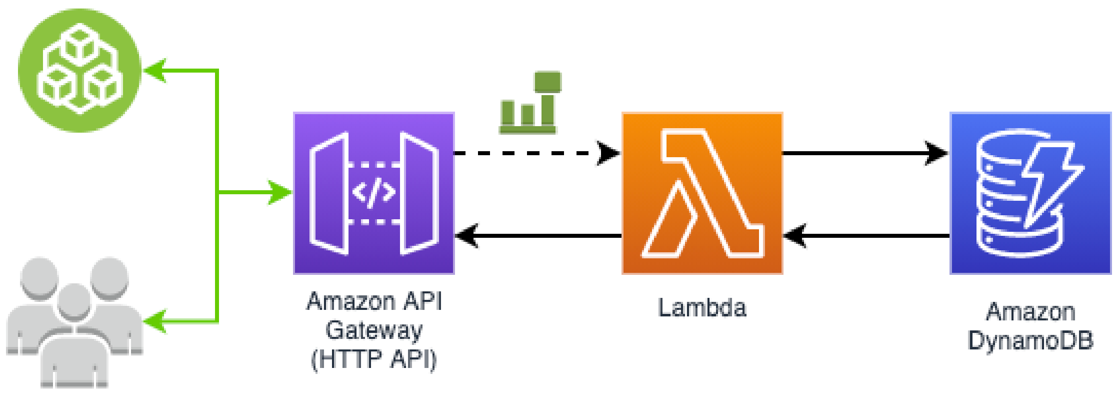
Infrastructure technologies

Technology	License	Notes
AWS	?	

Detailed Technical Design

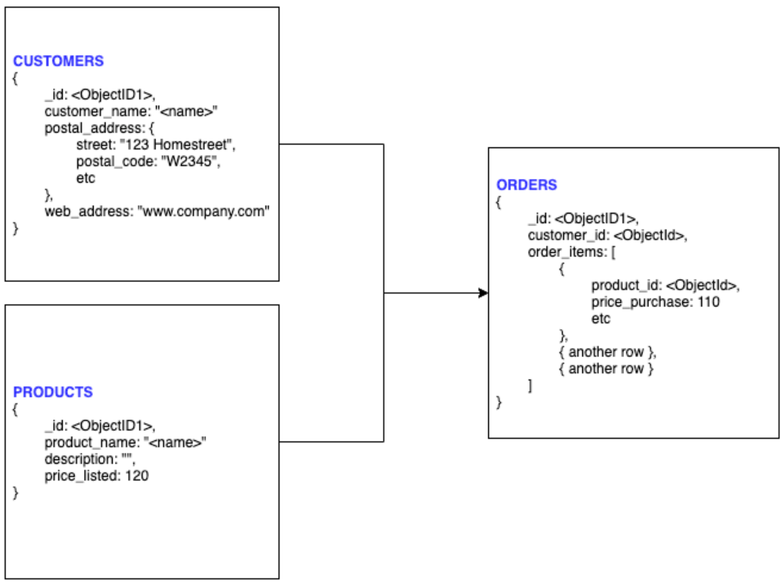
Application Architecture

This is the architecture overview and I propose that we use an API Gateway as provides in an easy and robust way with basic API technologies such as authentication as well and integrity against attacks such as DDOS. This stack also has the added benefit of being serverless



Step 1 - Database

I am assuming that there the data will be stored in a normalised data model in a DynamoDB similarly to this:



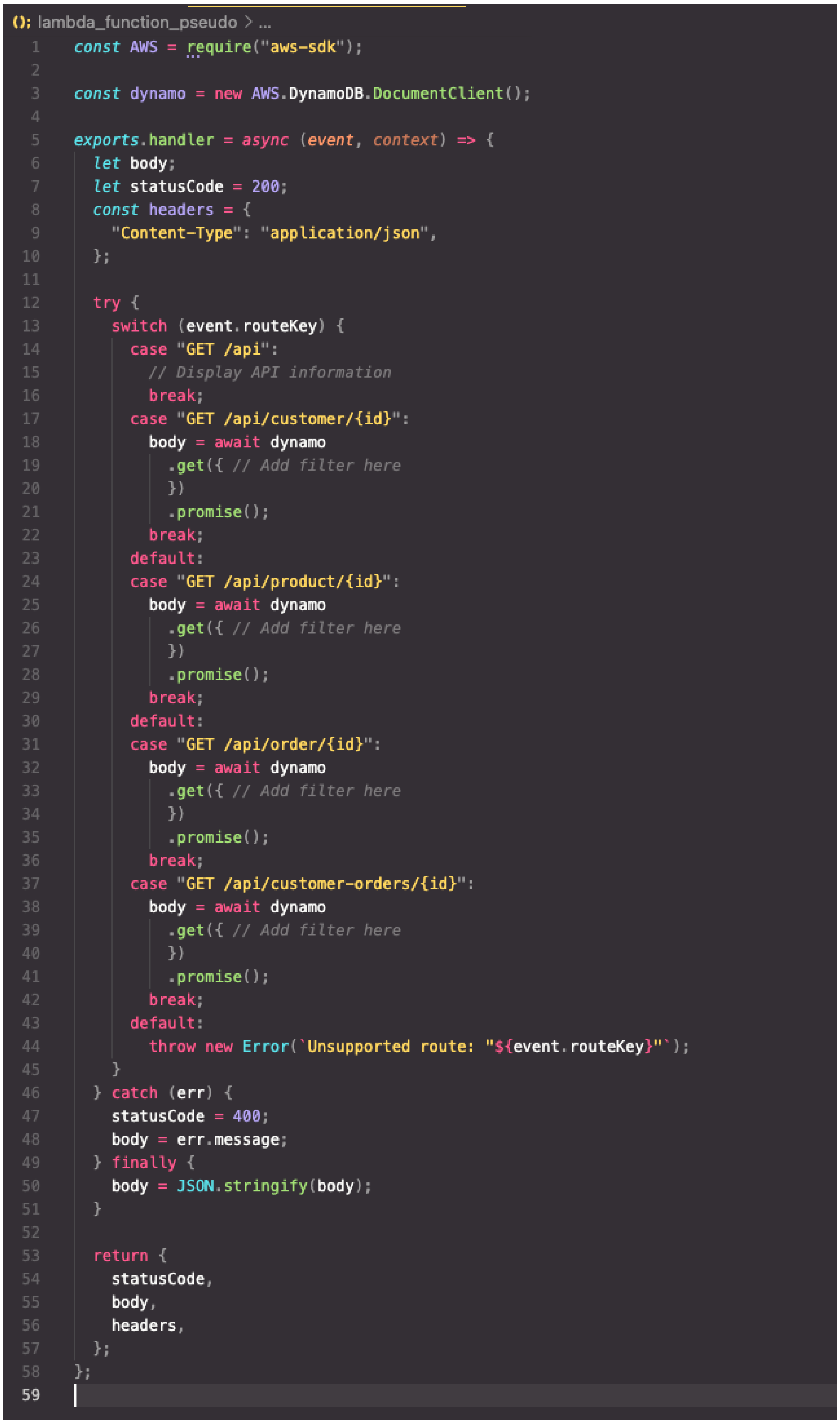
```
CUSTOMERS
{
        _id: <ObjectID1>,
        customer_name: "<name>"
        postal_address: {
                street: "123 Homestreet",
                postal_code: "W2345",
                etc
        },
        web_address: "www.company.com"
}
```

```
PRODUCTS
{
        _id: <ObjectID1>,
        product_name: "<name>"
        description: "",
        price_listed: 120
}
```

```
ORDERS
{
        _id: <ObjectID1>,
        customer_id: <ObjectId>,
        order_items: [
                {
                        product_id: <ObjectId>,
                        price_purchase: 110
                        etc
                },
                { another row },
                { another row }
        ]
}
```

Step 2 - Lambda function

The code providing the functionality will be in a Lambda function catching the routes that are to be set up in the API Gateway.



lambda_function_pseudo > ...

```javascript
const AWS = require("aws-sdk");

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

exports.handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  try {
    switch (event.routeKey) {
      case "GET /api":
        // Display API information
        break;
      case "GET /api/customer/{id}":
        body = await dynamo
          .get({ // Add filter here
          })
          .promise();
        break;
      default:
      case "GET /api/product/{id}":
        body = await dynamo
          .get({ // Add filter here
          })
          .promise();
        break;
      default:
      case "GET /api/order/{id}":
        body = await dynamo
          .get({ // Add filter here
          })
          .promise();
        break;
      case "GET /api/customer-orders/{id}":
        body = await dynamo
          .get({ // Add filter here
          })
          .promise();
        break;
      default:
        throw new Error(`Unsupported route: "${event.routeKey}"`);
    }
  } catch (err) {
    statusCode = 400;
    body = err.message;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};

```

Templated Endpoints

Path	Access	Details
`/api`	Public	Root path. Should not be used and could display information of how to gain access to the API
`/api/customer`	Authenticated	Endpoint for customer information
`/api/product`	Authenticated	Endpoint for product information
`/api/order`	Authenticated	Endpoint for order information
`/api.customer-orders`	Authenticated	Endpoint for customer orders

API Endpoints

Path	`/api/customer/{id}`
Access	Bearer Token
Methods	GET
Content-Type	application/json
Data Structure	
Details	The purpose of this endpoint is to give information about one specific customer

Field	Type	Notes
`id`	Integer	
`name`	String	Display name
`address`	Object	Customers address
`web`	String	Website address

Path	`/api/product/{id}`
Access	Bearer Token
Methods	GET
Content-Type	application/json
Data Structure	

Field	Type	Notes
`id`	Integer	
`name`	String	Display name of product
`price_listed`	Numeric	The list price of the product

Details	The purpose of this endpoint is to give information about one specific product

Path	`/api/order/{id}`
Access	Bearer Token
Methods	GET
Content-Type	application/json
Data Structure	

Field	Type	Notes
`id`	Integer	
`customer_name`	String	Display name of product
`order_items`	Array of Objects	The order items

Details	The purpose of this endpoint is to give information about one specific order

Path	`/api/customerorders/{id}`
Access	Bearer Token
Methods	GET
Content-Type	application/json
Data Structure	
Details	The purpose of this endpoint is to give information about one specific order

Field	Type	Notes
`id`	Integer	Customer ID
`customer_name`	String	Display name of product
`orders`	Array of Objects	An array of order IDs (could be more information if needed)

Step 3 – API Gateway configuration

The API gateway needs to be set up to meet the endpoint specified in the Lambda function and that is done in the console for the API gateway.

It is now needed to create the routes as specified in the endpoint above.

Next step is to set up the integrations needed to connect the routes to the lambda function.

Test at this stage

Security Considerations

Possible threat scenarios and the steps taken to mitigate them are described below.

A malicious user attempts a DOS (denial-of-service) attack.

This is handled by the Amazon API Gateway

A malicious user gains access to the secured instance.

The only functionality accessible is read only effectively hindering any data manipulation.

Document History

Author	Date	Notes
Patrik Oskarsson	2021-06-09	First version