Backend Design Document

1. Status

Work In Progress

2. Introduction

The document is responsible for backend design on simulators and service/data APIs. The backend is the main contact for the frontend to interact with any data and any external services. It will be completely serverless and require no virtual machines.

2.1 Glossary

Term	Definition
OEE	Overall Equipment Effectiveness is a metric used to evaluate how effectively a manufacturing operation utilizes its equipment, combining availability, performance, and quality
API	Application Programming Interface is a set of rules that specifies how software can communicate with different software systems.
UUID	UUID is a universally unique identifier

3. Requirements

Priority	Requirement	Done
0	Ability to define users, sites, machines(assets), simulator models, and OEE relationships	☐
0	See map data feed across a time range up to "now"	☐
0	Calculate OEE based on Availability, Performance, and Quality at the asset, site, and global level over a given time range	☐

0	Enable multiple free form dashboards to be created combining any available data or OEE calculations from the system	☐
1	Measure carbon footprint data to report on sustainability	☐
1	Support ad hoc Generative AI queries about production and received accurate answers, including calculated results and aggregated data	☐

4. Infrastructure

4.1. AWS Cloudformation

This service will provision and manage AWS resources.

4.11 Documentation

- AWS Cloudformation

4.2 AWS Lambda

This service provides serverless compute functions. These functions will interact with other AWS services They will be programmed in Go.

4.21 Documentation

- AWS Lambda for Go
- AWS SDK for Go
- AWS Cloudformation for Lambda

4.3 AWS API Gateway

This service will manage and monitor our API.

4.31 Documentation

- AWS Cloudformation for API Gateway

4.4 AWS DynamoDB

This service will be our serverless, key-value NoSQL database.

4.41 Documentation

- AWS Cloudformation for DynamoDB

4.5 AWS IAM

This service will manage identity and access roles for the other AWS services.

4.51 Documentation

- AWS Cloudformation for IAM

5. Database

This will outline the data models for our NoSQL database.

5.1 Factory

`Factory` represents a site.

Name	Type	Key Type	Example
factoryId	S - String	Partition Key	be656910-7a33-4439-abd1-f996245f6f1b
name	S - String	N/A	UH Factory
location	M - Map	N/A	{“longitude”: 95.3422, “latitude”: 29.7199}
description	S - String	N/A	This is the University of Houston factory.

5.11 location

Name	Type	Key Type	Example
longitude	N - Number	N/A	95.3422

latitude	N - Number	N/A	29.7199

5.2 Documentation

- AWS DynamoDB Data Types

6. API

This will outline the endpoints for our serverless API.
Base URL: https://api.example.com/v1

6.1 Factory

Path: /factories

Method	Description	Example
POST	CREATE a factory	POST /factories

6.11 Query Parameters

Parameter: ?id=

Method	Description	Example
GET	READ a factory's data	GET /factories?id=be656910-7a33-4439-abd1-f996245f6f1b
PUT	UPDATE a factory's data	PUT /factories?id=be656910-7a33-4439-abd1-f996245f6f1b
DELETE	DELETE a factory	DELETE /factories?id=be656910-7a33-4439-abd1-f996245f6f1b

7. Testing

This will outline any testing for the backend.

7.1. Factory

7.11 Unit Testing

7.111 Data Model Validation

Objective: Ensure that the structs representing the `Factory` entity correctly map to the expected DynamoDB data model
Test Cases:

- Validate struct tags for DynamoDB attributes.
- Test creation of a `Factory` instance with all required fields to ensure it matches the expected schema.

7.112 POST /factories

Objective: Test the function handling the POST request to create a new factory, ensuring it correctly processes valid input and handles invalid cases appropriately.
Test Cases:

- Success Case: Mock DynamoDB to simulate a successful insertion and verify the function's response and status code.
- Failure Case: Provide invalid input data to test validation logic and error handling.

7.113 GET /factories?id=

Objective: Verify that the function retrieves a factory's data correctly using a provided `factoryId`.
Test Cases:

- Existing Factory: Mock DynamoDB response to return a factory item for a valid `factoryId`.
- Non-Existing Factory: Simulate a DynamoDB response where the `factoryId` does not exist and verify handling.

7.114 PUT /factories?id=

Objective: Ensure the update functionality correctly modifies an existing factory's data.
Test Cases:

- Update Success: Mock a successful update scenario in DynamoDB and verify the function's response.
- Validation Failure: Test the endpoint with invalid update data to ensure proper error responses.

7.115 DELETE /factories?id=

Objective: Test the delete operation's ability to remove a factory entry from the database.
Test Cases:

- Delete Success: Mock DynamoDB to simulate successful deletion and verify the response.
- Non-Existing Factory: Attempt to delete a factory with an invalid `factoryId` and test error handling.

7.12 Integration Testing

7.121 Setup

- Installing and configuring DynamoDB Local
- Configure testing environment
- Deploy local serverless environment

7.122 POST /factories

Objective: Verify the endpoint correctly creates a new factory record in DynamoDB.
Setup: Ensure no existing factory record with the same ID.
Execution: Send a POST request with a valid factory JSON body.
Validation:

- Response status code is 201 Created.
- Response body includes a factory ID.
- Record is verifiably created in DynamoDB.

7.123 GET /factories?id=

Objective: Ensure the endpoint retrieves a factory's details correctly.
Setup: Create a test factory record in DynamoDB.
Execution: Send a GET request with the factory ID as a query parameter.
Validation:

- Response status code is 200 OK.
- Response body matches the factory's details.

7.124 PUT /factories?id=

Objective: Test the endpoint's ability to update an existing factory's details.
Setup: Create a test factory record in DynamoDB.
Execution: Send a PUT request with updated factory details and the factory ID as a query parameter.
Validation:

- Response status code is 200 OK.
- DynamoDB record reflects the updated details.

7.125 DELETE /factories?id=

Objective: Confirm the endpoint correctly deletes a factory record.
Setup: Create a test factory record in DynamoDB.
Execution: Send a DELETE request with the factory ID as a query parameter.
Validation:

- Response status code is 204 No Content.
- The factory record is no longer present in DynamoDB

8. Work Log

This will outline the tasks for the backend.

#	Task	Sprint	Status	Assignee
1	Setup AWS account	1	Completed	jaqmedina9@g… Bryant Le
2	Design document schema for Factory data	1	Completed	Bryant Le
3	Design API endpoints for Factory	1	Completed	Bryant Le
4	Setup AWS Cloudformation for Lambda	1	In Progress	
5	Setup AWS Cloudformation for API Gateway	1	In Progress	
6	Setup AWS Cloudformation for DynamoDB	1	In Progress	
7	Develop Go function for CREATE Factory	1	Completed	Bryant Le
8	Develop Go function for READ Factory	1	In Progress	
9	Develop Go function for UPDATE Factory	1	In Progress	
10	Develop Go function for DELETE Factory	1	In Progress	