Comprehensive AWS Implementation Guide for Logistics Management System

This enhanced guide provides extremely detailed instructions for setting up your logistics system on AWS. I'll walk through every step, field, button click, and configuration option to ensure nothing is missed.

1. AWS Account Setup

Step 1: AWS Account Login

1. Open your web browser and navigate to https://aws.amazon.com/
2. Click the "Sign In to the Console" button in the top-right corner
3. Enter your AWS account email address and password
4. Complete any multi-factor authentication if enabled
5. Once logged in, you'll see the AWS Management Console

Step 2: Select the Appropriate Region

1. In the top-right corner of the console, find the region dropdown (next to your account name)
2. Click the dropdown and select the region closest to your users (e.g., "US East (N. Virginia)" or "eu-west-1")
3. Note: All services you create must be in the same region to work together properly

2. Frontend Hosting with Amazon S3

Step 1: Create an S3 Bucket for Frontend Hosting

1. In the AWS Management Console, click the search bar at the top
2. Type "S3" and click on "S3" from the dropdown results
3. On the S3 dashboard, click the orange "Create bucket" button
4. In the "Create bucket" form:
    - **General configuration section**:
        - Bucket name: Type intellilogistics-frontend-[your-initials] (e.g., intellilogistics-frontend-tk)

        - AWS Region: Ensure it matches the region you selected earlier
    - **Object Ownership section**:
        - Select "ACLs disabled (recommended)"
    - **Block Public Access settings for this bucket section**:
        - Uncheck the box labeled "Block all public access"
        - A warning message will appear about making the bucket public
        - Check the acknowledgment box that says "I acknowledge that the current settings might result in this bucket and the objects within becoming public"
    - **Bucket Versioning section**:
        - Select "Disable" (or "Enable" if you want versioning)
    - **Tags section** (optional):
        - Add a tag with Key: "Project" and Value: "IntelliLogistics" if desired
    - **Default encryption section**:
        - Keep the default "Server-side encryption with Amazon S3 managed keys (SSE-S3)"
    - **Advanced settings section**:
        - Leave as default
5. Review all settings and click the orange "Create bucket" button at the bottom right
6. You should see a green success message and your new bucket in the list

Step 2: Configure the Bucket for Static Website Hosting

1. In the S3 bucket list, click on the name of your newly created bucket
2. In the bucket details page, click the "Properties" tab near the top
3. Scroll down until you find the "Static website hosting" card/section
4. Click the "Edit" button in that section
5. In the "Edit static website hosting" form:

    - Select "Enable" under "Static website hosting"
    - For "Hosting type", select "Host a static website"
    - In the "Index document" field, type index.html
    - In the "Error document" field, type index.html
    - Leave "Redirection rules" empty
6. Click the orange "Save changes" button
7. Scroll back to the "Static website hosting" section to see your website endpoint URL
8. Copy this URL and save it somewhere (e.g., Notepad) for later use. It will look something like http://intellilogistics-frontend-tk.s3-website-us-east-1.amazonaws.com

Step 3: Configure Bucket Permissions

1. Stay in your bucket details page and click the "Permissions" tab
2. Scroll down to the "Bucket policy" section and click "Edit"
3. In the policy editor text box, paste the following policy (replace intellilogistics-frontend-tk with your actual bucket name):

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::intellilogistics-frontend-tk/*"
    }
  ]
}
```

4. Double-check that you've replaced the bucket name with your actual bucket name
5. Click the orange "Save changes" button
6. You should see a warning message saying "This bucket has public access" which is expected

Step 4: Upload Frontend Files

1. Stay in your bucket details page and click the "Objects" tab
2. Click the orange "Upload" button
3. In the "Upload" page:
    - Click the "Add files" button to select all your HTML files (index.html, add-shipment.html, cost-estimation.html, feedback.html)
    - Click "Upload" (we'll add more files in subsequent uploads)
4. Repeat the upload process for your CSS files:
    - Click "Upload" button
    - Click "Add folder" button
    - Select your "css" folder containing style.css and responsive.css
    - Click "Upload"
5. Repeat again for your JavaScript files:
    - Click "Upload" button
    - Click "Add folder" button
    - Select your "js" folder containing all JavaScript files
    - Click "Upload"
6. If you have any image assets, repeat the process for those as well
7. After all uploads complete, you should see all your files and folders in the bucket

Step 5: Verify Your Static Website

1. Go back to the "Properties" tab
2. Scroll down to "Static website hosting" section

3. Click on the website endpoint URL (the one you saved earlier)
4. Your IntelliLogistics website should load in the browser
5. Test navigation between pages to ensure all links work
6. If you see any errors, check the browser's developer console (F12) for clues

3. Database Setup with Amazon DynamoDB

Step 1: Create the Shipments Table

1. In the AWS Management Console, click the search bar at the top
2. Type "DynamoDB" and click on "DynamoDB" from the dropdown results
3. On the DynamoDB dashboard, click the orange "Create table" button
4. In the "Create table" form:
   - **Table details section**:
     - Table name: Type Shipments
     - Partition key: Type shipmentId in the field
     - For the partition key data type, select "String" from the dropdown
   - **Table settings section**:
     - Select "Default settings" (this uses on-demand capacity mode)
   - Alternatively, if you want to configure specific settings:
     - Select "Customize settings"
     - For "Capacity mode", choose "Provisioned" (this is more cost-effective for predictable workloads)
     - Set Read capacity units (RCUs) to 5
     - Set Write capacity units (WCUs) to 5
     - Under "Secondary indexes", leave as is (none)
     - Under "Encryption at rest", leave as default (AWS owned key)
5. Review settings and click the orange "Create table" button

6. You'll see a notification that your table is being created
7. Wait until the table status shows "Active" before proceeding

Step 2: Create the Feedback Table

1. Still in the DynamoDB dashboard, click the orange "Create table" button
2. In the "Create table" form:
    - **Table details section**:
        - Table name: Type Feedback
        - Partition key: Type feedbackId in the field
        - For the partition key data type, select "String" from the dropdown
    - **Table settings section**:
        - Select "Default settings"
3. Click the orange "Create table" button
4. Wait until the table status shows "Active" before proceeding

Step 3: Create the Users Table (Optional)

1. Still in the DynamoDB dashboard, click the orange "Create table" button
2. In the "Create table" form:
    - **Table details section**:
        - Table name: Type Users
        - Partition key: Type userId in the field
        - For the partition key data type, select "String" from the dropdown
    - **Table settings section**:
        - Select "Default settings"
3. Click the orange "Create table" button
4. Wait until the table status shows "Active" before proceeding

4. Backend Setup with AWS Lambda

Step 1: Create IAM Role for Lambda Functions

1. In the AWS Management Console, click the search bar at the top
2. Type "IAM" and click on "IAM" from the dropdown results
3. In the left navigation menu, click on "Roles"
4. Click the blue "Create role" button
5. In the "Create role" wizard:
    - **Trusted entity type section**:
        - Select "AWS service"
    - **Use case section**:
        - Select "Lambda" under the "Common use cases" section
    - Click the "Next" button
6. On the "Add permissions" page:
    - In the search box, type "DynamoDB"
    - Check the box next to "AmazonDynamoDBFullAccess"
    - Clear the search box and type "Lambda"
    - Check the box next to "AWSLambdaBasicExecutionRole"
    - Click the "Next" button
7. On the "Name, review, and create" page:
    - In the "Role name" field, type LogisticsLambdaRole
    - In the "Description" field, type Role for Logistics Management Lambda Functions
    - Review the permissions
    - Click the "Create role" button
8. You should see a green success message

Step 2: Create the Create Shipment Lambda Function

1. In the AWS Management Console, click the search bar at the top
2. Type "Lambda" and click on "Lambda" from the dropdown results
3. Click the orange "Create function" button
4. In the "Create function" page:

    - Select "Author from scratch"
    - **Basic information section**:
        - Function name: Type createShipment
        - Runtime: Select "Python 3.9" from the dropdown
        - Architecture: Select "x86_64"
        - **Permissions section**:
            - Execution role: Select "Use an existing role"
            - Existing role: Select "LogisticsLambdaRole" from the dropdown
    - Advanced settings: Leave collapsed/default
    - Click the orange "Create function" button
5. On the function configuration page, you'll see the code editor
6. Replace the default code in the lambda_function.py file with the following:

Python

```
import json
import boto3
import uuid
import datetime
from decimal import Decimal

# Helper class to convert Decimal to float for JSON serialization
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, Decimal):
            return float(o)
        return super(DecimalEncoder, self).default(o)
```

```
# Initialize DynamoDB resource and table
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Shipments')

def lambda_handler(event, context):
    try:
        print("Received event:", json.dumps(event))

        # Parse body content from API Gateway
        if 'body' in event:
            if isinstance(event['body'], str):
                body = json.loads(event['body'])
            else:
                body = event['body']
        else:
            body = event

        print("Parsed body:", json.dumps(body))

        # Generate unique ID for the shipment
        shipment_id = 'SH-' + str(uuid.uuid4())[:8].upper()

        # Get current timestamp
        current_time = datetime.datetime.now().isoformat()

        # Add required fields
        shipment = {
            'shipmentId': shipment_id,
            'createdAt': current_time,
            'status': 'Processing',
            'customerName': body.get('customerName', ''),
            'customerEmail': body.get('customerEmail', ''),
            'customerPhone': body.get('customerPhone', ''),
            'customerCompany': body.get('customerCompany', ''),
            'originAddress': body.get('originAddress', ''),
            'originCity': body.get('originCity', ''),
            'originState': body.get('originState', ''),
            'originZip': body.get('originZip', ''),
            'originCountry': body.get('originCountry', ''),
            'destinationAddress': body.get('destinationAddress', ''),
            'destinationCity': body.get('destinationCity', ''),
            'destinationState': body.get('destinationState', ''),
            'destinationZip': body.get('destinationZip', ''),
            'destinationCountry': body.get('destinationCountry', ''),
            'shipmentDate': body.get('shipmentDate', ''),
            'deliveryDate': body.get('deliveryDate', ''),
            'shipmentType': body.get('shipmentType', ''),
            'transportMode': body.get('transportMode', ''),
            'packageType': body.get('packageType', ''),
            'packageQuantity': body.get('packageQuantity', ''),
            'weight': body.get('weight', ''),
            'dimensions': {
                'length': body.get('length', ''),
                'width': body.get('width', ''),
                'height': body.get('height', '')
            },
            'contents': body.get('contents', ''),
            'fragile': body.get('fragileCheck') == 'on',
            'hazardous': body.get('hazardousCheck') == 'on',
            'specialInstructions': body.get('specialInstructions', ''),
            'reference': body.get('reference', ''),
            'insurance': body.get('insurance', '')
        }

        # Convert empty strings to None for DynamoDB
        for key, value in shipment.items():
            if value == '':
                shipment[key] = None

        # Print what we're about to save
        print("Saving shipment:", json.dumps(shipment, cls=DecimalEncoder))

        # Store the shipment in DynamoDB
        response = table.put_item(Item=shipment)
        print("DynamoDB response:", response)

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps({
                'success': True,
                'message': 'Shipment created successfully',
                'shipmentId': shipment_id
            })
        }
    except Exception as e:
        print(f"Error: {str(e)}")
        import traceback
        traceback.print_exc()

        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps({
                'success': False,
                'message': f'Error creating shipment: {str(e)}'
            })
        }
```

7. Click the "Deploy" button to save your function

Step 3: Create the Get Shipments Lambda Function

1. Go back to the Lambda dashboard
2. Click the orange "Create function" button
3. In the "Create function" page:
    - Select "Author from scratch"
    - **Basic information section**:
        - Function name: Type getShipments
        - Runtime: Select "Python 3.9" from the dropdown
        - Architecture: Select "x86_64"
        - **Permissions section**:
            - Execution role: Select "Use an existing role"
            - Existing role: Select "LogisticsLambdaRole" from the dropdown
    - Click the orange "Create function" button
4. Replace the default code with:

Python

```python
import json
import boto3
from decimal import Decimal

# Helper class to convert Decimal to float for JSON serialization
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, Decimal):
            return float(o)
        return super(DecimalEncoder, self).default(o)

# Initialize DynamoDB resource and table
dynamodb = boto3.resource('dynamodb')
```

```
table = dynamodb.Table('Shipments')

def lambda_handler(event, context):
    try:
        print("Received event:", json.dumps(event))

        # Scan DynamoDB table to get all shipments
        response = table.scan()

        shipments = response.get('Items', [])

        # Continue scanning if we have more items (pagination)
        while 'LastEvaluatedKey' in response:
            response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
            shipments.extend(response.get('Items', []))

        # Sort by creation date (newest first)
        shipments.sort(key=lambda x: x.get('createdAt', ''), reverse=True)

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,GET'
            },
            'body': json.dumps({
                'success': True,
                'count': len(shipments),
                'shipments': shipments
            }, cls=DecimalEncoder)
        }
    except Exception as e:
        print(f"Error: {str(e)}")
        import traceback
        traceback.print_exc()

        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,GET'
            },
            'body': json.dumps({
                'success': False,
                'message': f'Error retrieving shipments: {str(e)}'
            })
        }
```

5. Click the "Deploy" button to save your function

Step 4: Create the Get Shipment by ID Lambda Function

1. Go back to the Lambda dashboard

2. Click the orange "Create function" button
3. In the "Create function" page:
    - Select "Author from scratch"
    - **Basic information section**:
        - Function name: Type getShipmentById
        - Runtime: Select "Python 3.9" from the dropdown
        - Architecture: Select "x86_64"
        - **Permissions section**:
            - Execution role: Select "Use an existing role"
            - Existing role: Select "LogisticsLambdaRole" from the dropdown
    - Click the orange "Create function" button
4. Replace the default code with:

Python

```
import json
import boto3
from decimal import Decimal

# Helper class to convert Decimal to float for JSON serialization
class DecimalEncoder(json.JSONEncoder):
   def default(self, o):
      if isinstance(o, Decimal):
         return float(o)
      return super(DecimalEncoder, self).default(o)

# Initialize DynamoDB resource and table
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Shipments')
```

```
def lambda_handler(event, context):
    try:
        print("Received event:", json.dumps(event))

        # Get the shipment ID from path parameters
        shipment_id = None

        # Extract shipmentId from different possible event structures
        if 'pathParameters' in event and event['pathParameters'] and 'shipmentId'
in event['pathParameters']:
            shipment_id = event['pathParameters']['shipmentId']
        elif 'queryStringParameters' in event and event['queryStringParameters']
and 'shipmentId' in event['queryStringParameters']:
            shipment_id = event['queryStringParameters']['shipmentId']

        if not shipment_id:
            return {
                'statusCode': 400,
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({
                    'success': False,
                    'message': 'Missing shipment ID'
                })
            }
```

```
# Get the shipment from DynamoDB
response = table.get_item(
    Key={
        'shipmentId': shipment_id
    }
)

# Check if the shipment exists
if 'Item' not in response:
    return {
        'statusCode': 404,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'success': False,
            'message': f'Shipment with ID {shipment_id} not found'
        })
    }

# Return the shipment
shipment = response['Item']

return {
    'statusCode': 200,
    'headers': {
```

```
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,GET'
            },
            'body': json.dumps({
                'success': True,
                'shipment': shipment
            }, cls=DecimalEncoder)
        }
    except Exception as e:
        print(f"Error: {str(e)}")
        import traceback
        traceback.print_exc()

        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,GET'
            },
            'body': json.dumps({
                'success': False,
                'message': f'Error retrieving shipment: {str(e)}'
    })
  }
```

5. Click the "Deploy" button to save your function

Step 5: Create the Submit Feedback Lambda Function

1. Go back to the Lambda dashboard
2. Click the orange "Create function" button
3. In the "Create function" page:
    - Select "Author from scratch"
    - **Basic information section**:
        - Function name: Type submitFeedback
        - Runtime: Select "Python 3.9" from the dropdown
        - Architecture: Select "x86_64"
        - **Permissions section**:
            - Execution role: Select "Use an existing role"
            - Existing role: Select "LogisticsLambdaRole" from the dropdown
    - Click the orange "Create function" button
4. Replace the default code with:

Python

```
import json
import boto3
import uuid
import datetime

# Initialize DynamoDB resource and table
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Feedback')
```

```
def lambda_handler(event, context):
    try:
        print("Received event:", json.dumps(event))

        # Parse body content from API Gateway
        if 'body' in event:
            if isinstance(event['body'], str):
                body = json.loads(event['body'])
            else:
                body = event['body']
        else:
            body = event

        print("Parsed body:", json.dumps(body))

        # Generate unique ID for the feedback
        feedback_id = 'FB-' + str(uuid.uuid4())[:8].upper()

        # Get current timestamp
        current_time = datetime.datetime.now().isoformat()

        # Create feedback item
        feedback = {
            'feedbackId': feedback_id,
            'createdAt': current_time,
            'name': body.get('name', ''),
            'email': body.get('email', ''),
            'feedbackType': body.get('feedbackType', ''),
            'subject': body.get('subject', ''),
            'message': body.get('message', ''),
            'shipmentId': body.get('shipmentId', ''),
            'priority': body.get('priority', 'medium'),
            'contactMethod': body.get('contactMethod', 'email')
        }

        # Convert empty strings to None for DynamoDB
        for key, value in feedback.items():
            if value == '':
                feedback[key] = None

        # Store the feedback in DynamoDB
        response = table.put_item(Item=feedback)
        print("DynamoDB response:", response)

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,POST'
            },
            'body': json.dumps({
                'success': True,
                'message': 'Feedback submitted successfully',
                'feedbackId': feedback_id
            })
        }
    except Exception as e:
        print(f"Error: {str(e)}")
        import traceback
        traceback.print_exc()

        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token',
                'Access-Control-Allow-Methods': 'OPTIONS,POST'
            },
            'body': json.dumps({
                'success': False,
                'message': f'Error submitting feedback: {str(e)}'
            })
        }
```

5. Click the "Deploy" button to save your function

5. API Gateway Setup

Step 1: Create a REST API

1. In the AWS Management Console, click the search bar at the top
2. Type "API Gateway" and click on "API Gateway" from the dropdown results

3. Click the orange "Create API" button
4. In the "Choose an API type" section, select "REST API" (not the private one) and click the orange "Build" button
5. In the "Create new API" section:
    - Select "New API"
    - For "API name", type LogisticsAPI
    - For "Description", type API for Logistics Management System
    - For "Endpoint Type", select "Regional"
    - Click the blue "Create API" button

Step 2: Create the Shipments Resource and POST Method

1. In your new API, in the "Resources" section, you should see "/" as the root resource
2. Click "Actions" dropdown button and select "Create Resource"
3. In the "New Resource" form:
    - Select "Resource Path" as "/"
    - For "Resource Name", type shipments
    - Leave "Resource Path" as "/shipments"
    - Leave "Enable API Gateway CORS" unchecked for now
    - Click "Create Resource"
4. With your new "/shipments" resource selected, click "Actions" dropdown and select "Create Method"
5. A small dropdown will appear under the resource
6. Select "POST" from the dropdown, then click the checkmark button
7. In the "Setup" form:
    - For "Integration type", select "Lambda Function"
    - Check "Use Lambda Proxy integration"
    - For "Lambda Region", select your region
    - For "Lambda Function", type createShipment and select it from the dropdown

   - Click "Save"
8. A popup will appear asking to give API Gateway permission to invoke your Lambda function - click "OK"

Step 3: Create the GET Method for Shipments

1. With your "/shipments" resource still selected, click "Actions" dropdown and select "Create Method"
2. Select "GET" from the dropdown, then click the checkmark button
3. In the "Setup" form:
   - For "Integration type", select "Lambda Function"
   - Check "Use Lambda Proxy integration"
   - For "Lambda Region", select your region
   - For "Lambda Function", type getShipments and select it from the dropdown
   - Click "Save"
4. Click "OK" in the permission popup

Step 4: Create a Shipment By ID Resource and GET Method

1. With the "/shipments" resource selected, click "Actions" dropdown and select "Create Resource"
2. In the "New Resource" form:
   - For "Resource Path", ensure "/shipments" is selected
   - For "Resource Name", type {shipmentId}
   - For "Resource Path", it should show "/shipments/{shipmentId}"
   - Leave "Enable API Gateway CORS" unchecked
   - Click "Create Resource"
3. With your new "/shipments/{shipmentId}" resource selected, click "Actions" dropdown and select "Create Method"
4. Select "GET" from the dropdown, then click the checkmark button
5. In the "Setup" form:
   - For "Integration type", select "Lambda Function"

- Check "Use Lambda Proxy integration"
- For "Lambda Region", select your region
- For "Lambda Function", type getShipmentById and select it from the dropdown
- Click "Save"

6. Click "OK" in the permission popup

Step 5: Create the Feedback Resource and POST Method

1. Go back to the root resource ("/") by clicking on it in the resource tree
2. Click "Actions" dropdown and select "Create Resource"
3. In the "New Resource" form:
   - For "Resource Path", ensure "/" is selected
   - For "Resource Name", type feedback
   - Leave "Resource Path" as "/feedback"
   - Leave "Enable API Gateway CORS" unchecked
   - Click "Create Resource"
4. With your new "/feedback" resource selected, click "Actions" dropdown and select "Create Method"
5. Select "POST" from the dropdown, then click the checkmark button
6. In the "Setup" form:
   - For "Integration type", select "Lambda Function"
   - Check "Use Lambda Proxy integration"
   - For "Lambda Region", select your region
   - For "Lambda Function", type submitFeedback and select it from the dropdown
   - Click "Save"
7. Click "OK" in the permission popup

Step 6: Enable CORS for All Resources

1. Click on the "/shipments" resource
2. Click "Actions" dropdown and select "Enable CORS"

3. In the "Enable CORS" form:
   - For "Access-Control-Allow-Headers", enter 'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'
   - For "Access-Control-Allow-Methods", ensure all methods are selected
   - For "Access-Control-Allow-Origin", enter '*' (for production, use your specific domain)
   - Click "Enable CORS and replace existing CORS headers"
   - Click "Yes, replace existing values"
4. Do the same for the "/shipments/{shipmentId}" resource
5. Do the same for the "/feedback" resource

Step 7: Deploy the API

1. Click "Actions" dropdown and select "Deploy API"
2. In the "Deploy API" form:
   - For "Deployment stage", select "[New Stage]"
   - For "Stage name", type prod
   - For "Stage description", type Production environment
   - For "Deployment description", type Initial deployment
   - Click "Deploy"
3. You'll be taken to the "Stages" section
4. Expand the "prod" stage and select the root resource "/"
5. Copy the "Invoke URL" from the top of the page - it should look like https://abcd1234.execute-api.us-east-1.amazonaws.com/prod
6. Save this URL - this is your API endpoint root

6. Update Frontend Code to Connect to API

Step 1: Modify the shipment.js File

1. Open your local copy of the js/shipment.js file
2. Find the setupFormSubmission function

3. Replace it with the following (update the apiUrl with your actual API Gateway endpoint):

JavaScript

```
function setupFormSubmission() {
   const form = document.getElementById('shipmentForm');
   if (!form) return;

   form.addEventListener('submit', function(event) {
      event.preventDefault();

      // Show loading indicator
      const submitBtn = this.querySelector('button[type="submit"]');
      const originalBtnText = submitBtn.innerHTML;
      submitBtn.disabled = true;
      submitBtn.innerHTML = '<i class="fas fa-spinner fa-spin me-1"></i>
Creating...';

      // Collect form data
      const formData = new FormData(this);
      const shipmentData = {};
      for (const [key, value] of formData.entries()) {
         shipmentData[key] = value;
      }

      // API Gateway endpoint URL - REPLACE THIS WITH YOUR ACTUAL
ENDPOINT
      const apiUrl = 'https://abcd1234.execute-api.us-east-
1.amazonaws.com/prod/shipments';
```

```
        // Submit data to API
        fetch(apiUrl, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(shipmentData),
        })
        .then(response => {
          if (!response.ok) {
            throw new Error('Network response was not ok');
          }
          return response.json();
        })
        .then(data => {
          // Reset button
          submitBtn.disabled = false;
          submitBtn.innerHTML = originalBtnText;

          // Show success notification
          showNotification(`Shipment created successfully! Tracking ID:
${data.shipmentId}`, 'success');

          // Reset form after success
          form.reset();

          // Hide summary details
```

```
            document.getElementById('summaryContent')?.classList.remove('d-
none');
            document.getElementById('summaryDetails')?.classList.add('d-none');

            // Create a success modal
            showSuccessModal(data.shipmentId);
        })
        .catch(error => {
            // Reset button
            submitBtn.disabled = false;
            submitBtn.innerHTML = originalBtnText;

            // Show error notification
            showNotification('Error creating shipment. Please try again.', 'error');
            console.error('Error:', error);
        });
    });
}
```

Step 2: Modify the dashboard.js File

1. Open your local copy of the js/dashboard.js file
2. Add the following function to fetch shipments from your API:

JavaScript

```
function fetchShipments() {
    // API Gateway endpoint URL - REPLACE THIS WITH YOUR ACTUAL ENDPOINT
    const apiUrl = 'https://abcd1234.execute-api.us-east-
1.amazonaws.com/prod/shipments';

    // Show loading indicator
    const tableBody = document.getElementById('recentShipments');
    if (tableBody) {
        tableBody.innerHTML = '<tr><td colspan="7" class="text-center">Loading 
shipments...</td></tr>';
    }

    // Fetch shipments from API
    fetch(apiUrl)
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .then(data => {
        updateShipmentTable(data.shipments || []);
    })
    .catch(error => {
        console.error('Error:', error);
        if (tableBody) {
            tableBody.innerHTML = '<tr><td colspan="7" class="text-center text-
danger">Error loading shipments</td></tr>';
        }
    });
}

function updateShipmentTable(shipments) {
    const tableBody = document.getElementById('recentShipments');
```

```
  if (!tableBody) return;

  // Limit to 5 most recent shipments
  const recentShipments = shipments.slice(0, 5);

  if (recentShipments.length === 0) {
    tableBody.innerHTML = '<tr><td colspan="7" class="text-center">No
shipments found</td></tr>';
    return;
  }

  // Clear existing rows
  tableBody.innerHTML = '';

  // Add shipment rows
  recentShipments.forEach(shipment => {
    const row = document.createElement('tr');

    // Format delivery date
    let deliveryDate = 'Not specified';
    if (shipment.deliveryDate) {
      deliveryDate = new
Date(shipment.deliveryDate).toLocaleDateString('en-US', {
        month: 'short',
        day: 'numeric',
        year: 'numeric'
      });
    }
```

```
        // Create badge based on status
        let statusBadge = '';
        switch(shipment.status) {
            case 'Processing':
                statusBadge = '<span class="badge bg-warning text-
dark">Processing</span>';
                break;
            case 'In Transit':
                statusBadge = '<span class="badge bg-info">In Transit</span>';
                break;
            case 'Delivered':
                statusBadge = '<span class="badge bg-success">Delivered</span>';
                break;
            case 'Delayed':
                statusBadge = '<span class="badge bg-danger">Delayed</span>';
                break;
            default:
                statusBadge = `<span class="badge bg-
secondary">${shipment.status}</span>`;
        }

        // Build the row HTML
        row.innerHTML = `
            <td>${shipment.shipmentId}</td>
            <td>${shipment.customerName || 'N/A'}</td>
            <td>${shipment.originCity || ''}, ${shipment.originState || ''}</td>
            <td>${shipment.destinationCity || ''}, ${shipment.destinationState ||
''}</td>
```

```
            <td>${statusBadge}</td>
            <td>${deliveryDate}</td>
            <td>
                <button class="btn btn-sm btn-outline-primary"
onclick="viewShipment('${shipment.shipmentId}')">
                    <i class="fas fa-eye"></i>
                </button>
                <button class="btn btn-sm btn-outline-secondary"
onclick="editShipment('${shipment.shipmentId}')">
                    <i class="fas fa-edit"></i>
                </button>
            </td>
        `;

        tableBody.appendChild(row);
    });
}

// Function to view shipment details
function viewShipment(shipmentId) {
    // API Gateway endpoint URL - REPLACE THIS WITH YOUR ACTUAL ENDPOINT
    const apiUrl = `https://abcd1234.execute-api.us-east-
1.amazonaws.com/prod/shipments/${shipmentId}`;

    showNotification(`Loading shipment ${shipmentId}...`, 'info');

    fetch(apiUrl)
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .then(data => {
        // In a real app, you would show this data in a modal
        console.log('Shipment details:', data.shipment);
        showNotification(`Viewing details for shipment ${shipmentId}`, 'success');
    })
    .catch(error => {
        console.error('Error:', error);
        showNotification('Error loading shipment details', 'error');
    });
}

// Call this function when the dashboard loads
document.addEventListener('DOMContentLoaded', function() {
    // Existing code...

    // Add this line to fetch shipments when the page loads
    fetchShipments();
});
```

Step 3: Modify the feedback.js File

1. Open your local copy of the js/feedback.js file
2. Find the setupFeedbackFormSubmission function
3. Replace it with the following (update the apiUrl with your actual API Gateway endpoint):

JavaScript

```
function setupFeedbackFormSubmission() {
    const form = document.getElementById('feedbackForm');
    if (!form) return;

    form.addEventListener('submit', function(event) {
        event.preventDefault();

        // Validate all required fields
        let isValid = true;
        const requiredFields = form.querySelectorAll('[required]');

        requiredFields.forEach(field => {
            if (!validateField(field)) {
                isValid = false;
            }
        });

        // Validate file upload if present
        const fileInput = document.getElementById('screenshot');
        if (fileInput && fileInput.files.length && !validateFileUpload(fileInput)) {
            isValid = false;
        }

        // If not valid, stop submission
        if (!isValid) {
            showNotification('Please correct the errors in the form', 'error');
            return;
        }
        // Show loading state
        const submitBtn = form.querySelector('button[type="submit"]');
        const originalBtnText = submitBtn.innerHTML;
        submitBtn.disabled = true;
        submitBtn.innerHTML = '<i class="fas fa-spinner fa-spin me-1"></i>
Submitting...';

        // Collect form data
        const formData = new FormData(form);
        const feedbackData = {};
        for (const [key, value] of formData.entries()) {
          // Skip file inputs for now (would need separate file upload handling)
          if (key !== 'screenshot') {
            feedbackData[key] = value;
          }
        }

        // API Gateway endpoint URL - REPLACE THIS WITH YOUR ACTUAL
ENDPOINT
        const apiUrl = 'https://abcd1234.execute-api.us-east-
1.amazonaws.com/prod/feedback';

        // Submit data to API
        fetch(apiUrl, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(feedbackData),
        })
        .then(response => {
          if (!response.ok) {
            throw new Error('Network response was not ok');
          }
          return response.json();
        })
        .then(data => {
          // Reset button
          submitBtn.disabled = false;
          submitBtn.innerHTML = originalBtnText;

          // Show success and reset form
          showFeedbackSuccessMessage(data.feedbackId);
          form.reset();

          // Remove validation classes
          form.querySelectorAll('.is-valid, .is-invalid').forEach(el => {
            el.classList.remove('is-valid', 'is-invalid');
          });
        })
        .catch(error => {
          // Reset button
          submitBtn.disabled = false;
          submitBtn.innerHTML = originalBtnText;

          // Show error notification
```

```
            showNotification('Error submitting feedback. Please try again.', 'error');
            console.error('Error:', error);
        });
    });
}

// Update the success message function to use the actual feedback ID
function showFeedbackSuccessMessage(feedbackId) {
    // Create modal for success message
    const modalHtml = `
        <div class="modal fade" id="feedbackSuccessModal" tabindex="-1" aria-
hidden="true">
            <div class="modal-dialog modal-dialog-centered">
                <div class="modal-content">
                    <div class="modal-body text-center py-4">
                        <div class="mb-3">
                            <i class="fas fa-check-circle text-success fa-4x"></i>
                        </div>
                        <h3 class="modal-title mb-3">Thank You for Your Feedback!</h3>
                        <p class="mb-4">We appreciate you taking the time to share your
thoughts with us. Your feedback has been submitted successfully.</p>
                        <p class="text-muted">Reference ID: ${feedbackId}</p>
                    </div>
                    <div class="modal-footer border-0 justify-content-center pt-0">
                        <button type="button" class="btn btn-primary px-4" data-bs-
dismiss="modal">Close</button>
                    </div>
                </div>
            </div>
        </div>
    `;

    // Append modal to body
    document.body.insertAdjacentHTML('beforeend', modalHtml);

    // Initialize and show the modal
    const modal = new
bootstrap.Modal(document.getElementById('feedbackSuccessModal'));
    modal.show();

    // Remove modal from DOM when hidden

document.getElementById('feedbackSuccessModal').addEventListener('hidden.
bs.modal', function() {
        this.remove();
    });
}
```

Step 4: Re-upload the Updated JavaScript Files to S3

1. In the AWS Management Console, go to S3
2. Navigate to your intellilogistics-frontend bucket
3. Navigate to the js folder
4. Click "Upload"
5. Click "Add files"
6. Select your updated JS files (shipment.js, dashboard.js, feedback.js)
7. Click "Upload"
8. Wait for the upload to complete

7. Testing the Complete System

Step 1: Test the Create Shipment Functionality

1. Open your S3 website URL in your browser
2. Navigate to the "Add Shipment" page
3. Fill out the shipment form with test data
4. Click "Create Shipment"
5. Verify that you get a success message with a shipment ID

Step 2: Check Data in DynamoDB

1. In the AWS Management Console, go to DynamoDB
2. Click on "Tables" in the left navigation
3. Click on your "Shipments" table
4. Click on "Explore table items" button
5. You should see your newly created shipment in the table
6. Verify that all the fields match what you entered in the form

Step 3: Test the Dashboard Functionality

1. Go back to your website and navigate to the Dashboard
2. The recent shipments table should now show the shipment you just created
3. Try clicking the "View" button (eye icon) for your shipment
4. Verify that you see the notification about viewing the shipment details

Step 4: Test the Feedback Functionality

1. Navigate to the "Feedback" page
2. Fill out the feedback form
3. Click "Submit Feedback"
4. Verify that you get a success message with a feedback ID
5. Check the "Feedback" table in DynamoDB to ensure your feedback was saved

Step 5: Test Error Handling

1. Try submitting an empty form to verify validation works
2. Try refreshing the page to make sure the API calls still work

8. Troubleshooting Common Issues

API Gateway CORS Issues

If you see CORS errors in the browser console:

1. Go to API Gateway in the AWS Console
2. Select your API
3. For each resource, select "Actions" > "Enable CORS"
4. Ensure "Access-Control-Allow-Origin" is set to '*'
5. Click "Enable CORS and replace existing CORS headers"
6. Redeploy your API (Actions > Deploy API)

Lambda Function Errors

If your Lambda functions aren't working:

1. Go to the Lambda console
2. Select your function
3. Click on the "Monitor" tab
4. Click "View logs in CloudWatch"
5. Check the logs for error messages
6. Common issues:
   - Missing permissions to access DynamoDB
   - Incorrect table names
   - Incorrect parsing of event data

S3 Website Access Issues

If your S3 website isn't accessible:

1. Verify the bucket policy allows public access
2. Check that static website hosting is enabled
3. Make sure the bucket name in the policy matches your actual bucket name
4. Ensure all files were uploaded correctly

DynamoDB Issues

If data isn't appearing in DynamoDB:

1. Verify the Lambda function has permissions to write to DynamoDB
2. Check that the table name in the Lambda code matches your actual table name
3. Verify the partition key is set correctly