# Design Document

# The benefits are:

Knowledge Transfer:

* **Mitigate the "Bus Factor":** Ensures work continuity if a key member is unavailable.
* **Onboarding New Members:** Provides essential background for new team members.

Learning and Development

* **Educational Resource:** Serves as a learning tool for all team members.
* **Mentorship Opportunity:** Allows senior engineers to mentor less experienced team members.

Clarity and Direction

* **Clear Plan:** Reduces ambiguity with a well-defined project plan.
* **Defined Scope:** Manages expectations.
* **Collaborative Input:** Encourages team input, addressing ideas and concerns early.

Documentation

* **Reference Material**
* **Historical Record:** Keeps a record of decisions, scope changes, and design rationale.

Feedback Loop

* **Early Feedback:** Enables early feedback from stakeholders and team members.
* **Risk Mitigation:** Identifies potential risks and strategies early in the process.

# The Design Document Template:

**Section 1 - Project Description**

1.1 Project

Dog Breed Detector

1.2 Description

The **Dog Breed Detector** is a web-based application that allows users to upload an image of a dog and receive confident breed predictions by filtering data from AWS Rekognition. Alongside breed names, the app also provides fun bonus facts to educate the user.

**Section 2 - Overview**

2.1 Purpose

To make an easy-to-use, trustworthy application, which explains breed history for dogs of a mixed nature. This allows for easy background checks for rescued dogs and individuals looking to adopt.

2.2 Scope

Create an image generator that responds with all the data analyzed and collected using AWS services such as Rekognition, S3 and IAM.

**Section 3: System Architecture**

This app utilizes the following services:

**Frontend**: HTML + CSS + JavaScript

**Backend**: AWS Lambda (Python)

**Image Processing**: AWS Rekognition

**Storage**: S3 Bucket (for breed facts JSON file)

**API Management**: AWS API Gateway

The user uploads a dog image, which is sent to the API Gateway, invoking a Lambda function. The function uses Rekognition to analyze the image and detect breed labels, then cross-references breed names with a JSON file hosted in S3 to retrieve a fun fact for each detected breed (if available). All detected breeds with a confidence above 50% are displayed, to prevent false positives and enable more accurate info.

**Section 4: Data Dictionary**

| Field | Type | Description

|----------------|----------|------------------------------------------

| image | string | Base64-encoded dog image

| breed | string | Name of detected dog breed

| confidence | float | Confidence score from Rekognition

| fun\_fact | string | Breed-specific trivia (if available)

**Section 5 – Data Design**

Describe the data contained in databases and other shared structures between domains or within the scope of the overall project architecture

5.1 Persistent/Static Data

**S3 JSON File**: A structured object where each key is a dog breed and each value is a corresponding bonus fact.

Example:

{

"Labrador Retriever": "Labradors are gentle and great with families.",

"German Shepherd": "They are loyal and often serve as police or service dogs.",

...

}

-**Rekognition Response**: Parsed to collect only label names with confidence > 50%.

---

**Section 6 - User Interface Design**

- Light blue background

- White centered container with shadow and rounded edges

- Bone emoji border on top and bottom

- Cookie emoji border on left and right

-image upload button and detect breed button

- Image preview and breed predictions with fun facts

- Responsive layout

**Section 7 - Testing**

|  |  |
| --- | --- |
| **Objective** | Validate that the API accurately detects dog breeds and returns bonus facts and ensure that the UI displays correct results under different inputs. Also, if console is free from errors. |
| **Scope** | Focuses on breed detection accuracy, API response handling, S3 integration for facts, and UI display. Does **not** include testing on mobile devices. |
| **Resources** | AWS Console (Lambda, S3, API Gateway, Cloudwatch), Postman, VS Code (Live Server), Microsoft Edge |
| **Schedule** | Testing completed manually in stages between Mar 25–26 |
| **Test Cases** | See tables below for different test environments |

**Test Environment: Lambda + API Gateway**

| **Test Case** | **Input** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- |
| Valid dog image (Husky) | Base64 dog image of a husky | Breed: Husky + fun fact | ✅ Correct |
| Dog image (multiple breeds) | Dog with German Shepherd & Husky | Both breeds listed, each with fun facts | ❌ Only 1 shown initially (now fixed) |
| Invalid input | Empty image | Error message | ✅ Error handled |
| Unknown breed (not in JSON) | Akita photo | "No fact available" | ✅ Shown |

**UAT Environment: Front-End Integration**

| **Test Case** | **Input** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- |
| Upload & click "Detect" | Husky image | Image preview, breed name, confidence, fun fact | ✅ Successful |
| Upload non-dog image | Random object | No recognized breeds | ✅ Works |
| CORS & API Test | API POST call | Should complete with no fetch/CORS error | ✅ Fixed via OPTIONS method |

**Section 8 - Monitoring**

Monitoring is handled through **AWS CloudWatch** logs and status codes for Lambda executions.

| **Metric Type** | **Description** |
| --- | --- |
| **Performance Metrics** | Tracked by API latency (visible in CloudWatch metrics); most detections occur in under 2 seconds |
| **Error Metrics** | All Lambda exceptions are logged via CloudWatch; errors like bad input, missing image, or JSON parse issues are handled correctly |
| **Availability Metrics** | Uptime relies on AWS API Gateway and Lambda availability; currently deployed on a public /prod stage |
| **User Metrics** | User interaction is measured via console logs and manual observations |