```
public static class AnalyzeForm
   {
      private static readonly string formsRecognizerApiEndpointSetting =
"FORMS_RECOGNIZER_ENDPOINT_URL";
      private static readonly string formsRecognizerApiKeySetting = "FORMS_RECOGNIZER_API_KEY";
      private static readonly string modelIdSetting = "FORMS_RECOGNIZER_MODEL_ID";
      private static readonly string retryDelaySetting = "FORMS_RECOGNIZER_DELAY_ATTEMPTS";
      private static readonly string maxAttemptsSetting = "FORMS_RECOGNIZER_MAX_ATTEMPTS";

      [FunctionName("analyze-form")]
      public static async Task<IActionResult> RunAnalyzeForm(
         [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] HttpRequest req,
         ILogger log,
         ExecutionContext executionContext)
      {
         log.LogInformation("Analyze Form Custom Skill: C# HTTP trigger function processed a request.");

         string skillName = executionContext.FunctionName;
         IEnumerable<WebApiRequestRecord> requestRecords =
WebApiSkillHelpers.GetRequestRecords(req);
         if (requestRecords == null)
         {
            return new BadRequestObjectResult($"{skillName} - Invalid request record array.");
         }

         string formsRecognizerEndpointUrl =
Environment.GetEnvironmentVariable(formsRecognizerApiEndpointSetting,
EnvironmentVariableTarget.Process).TrimEnd('/');
         string formsRecognizerApiKey =
Environment.GetEnvironmentVariable(formsRecognizerApiKeySetting,
EnvironmentVariableTarget.Process);
```

```
        string modelId = Environment.GetEnvironmentVariable(modelIdSetting,
EnvironmentVariableTarget.Process);

        int retryDelay = int.TryParse(Environment.GetEnvironmentVariable(retryDelaySetting,
EnvironmentVariableTarget.Process), out int parsedRetryDelay) ? parsedRetryDelay : 1000;

        int maxAttempts = int.TryParse(Environment.GetEnvironmentVariable(maxAttemptsSetting,
EnvironmentVariableTarget.Process), out int parsedMaxAttempts) ? parsedMaxAttempts : 100;


        Dictionary<string, string> fieldMappings = JsonConvert.DeserializeObject<Dictionary<string,
string>>(
            File.ReadAllText($"{executionContext.FunctionAppDirectory}\\field-mappings.json"));


        WebApiSkillResponse response = await
WebApiSkillHelpers.ProcessRequestRecordsAsync(skillName, requestRecords,
            async (inRecord, outRecord) => {
                var formUrl = inRecord.Data["formUrl"] as string;
                var formSasToken = inRecord.Data["formSasToken"] as string;


                // Create the job
                string jobId = await GetJobId(formsRecognizerEndpointUrl, formUrl + formSasToken,
modelId, formsRecognizerApiKey);


                // Get the results
                for (int attempt = 0; attempt < maxAttempts; attempt++)
                {
                    (string status, JToken result) = await GetJobStatus(jobId, formsRecognizerApiKey);
                    if (status == "failed")
                    {
                        var errors = result.SelectToken("analyzeResult.errors") as JArray;
                        outRecord.Errors.AddRange(errors.Children().Select(error => new
WebApiErrorWarningContract
                        {
                            Message = error.SelectToken("message").ToObject<string>()
```

```
                }));
                return outRecord;
            }
            if (status == "succeeded")
            {
                List<Page> pages =
result.SelectToken("analyzeResult.pageResults").ToObject<List<Page>>();
                //List<Page> pages =
result.SelectToken("analyzeResult.documentResults[0]").ToObject<List<Page>>();
                foreach (KeyValuePair<string, string> kvp in fieldMappings)
                {
                    List<JToken> newresults = FindTokens(result, kvp.Key);
                    string value = "";
                    if (newresults.Count>0)
                    {
                        value = GetJTokenValue(newresults);
                    }
                    //string value = GetField(pages, kvp.Key);
                    if (!string.IsNullOrWhiteSpace(value))
                    {
                        outRecord.Data[kvp.Value] = value;
                    }
                }
                return outRecord;
            }
            await Task.Delay(retryDelay);
        }
        outRecord.Errors.Add(new WebApiErrorWarningContract
        {
            Message = $"The forms recognizer did not finish the job after {maxAttempts} attempts."
```

```
            });
            return outRecord;
        });

    return new OkObjectResult(response);
}

private static string GetJTokenValue(List<JToken> newresults)
{
    string value = newresults[0].SelectToken("valueString").ToString();
    return value;
}

/// <summary>
/// Searches for a field in a given page and returns the concatenated results.
/// </summary>
/// <param name="response">the responsed from the forms recognizer service.</param>
/// <param name="fieldName">The field to search for</param>
/// <returns></returns>
/*private static string GetField(IList<Page> pages, string fieldName)
{
    IEnumerable<string> value = pages
        .SelectMany(p => p.KeyValuePairs)
        .Where(kvp => string.Equals(kvp.Key.Text.Trim(), fieldName,
StringComparison.CurrentCultureIgnoreCase))
        .Select(kvp => kvp.Value.Text);
    return value == null ? null : string.Join(" ", value);
}*/
```

```
    /// <summary>
    /// Creates the analysis job and returns a job id
    /// </summary>
    /// <param name="documentBytes">The binary contents of the document to analyze.</param>
    /// <param name="modelId">The id of the trained model to use.</param>
    /// <returns>The job id that can be used in analyzeResults.</returns>
    private static async Task<string> GetJobId(string endpointUrl, string formUrl, string modelId, string apiKey)
    {
        string uri = endpointUrl + "/formrecognizer/v2.0-preview/custom/models/" + Uri.EscapeDataString(modelId) + "/analyze";

        using (var client = new HttpClient())
        {
            using (var request = new HttpRequestMessage
            {
                Method = HttpMethod.Post,
                RequestUri = new Uri(uri),
                Content = new StringContent(JsonConvert.SerializeObject(new
                {
                    source = formUrl
                })),
            })
            {
                request.Headers.Add("Ocp-Apim-Subscription-Key", apiKey);
                request.Content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
                using (HttpResponseMessage response = await client.SendAsync(request))
                {

                    if (response.StatusCode != HttpStatusCode.Accepted)
```

```
                {
                    throw new HttpRequestException($"The remote service {uri} responded with a {response.StatusCode} error code instead of the expected 202 Accepted.");
                }
                return response.Headers.GetValues("Operation-Location").FirstOrDefault();
            }
        }
    }
}

/// <summary>
/// Makes a request to get the job status, and the full response if the job's complete.
/// </summary>
/// <param name="modelId">The trained model id.</param>
/// <param name="jobId">The job id.</param>
/// <returns></returns>
private static async Task<(string status, JToken results)> GetJobStatus(string jobId, string apiKey)
{
    using (var client = new HttpClient())
    {
        using (var request = new HttpRequestMessage
        {
            Method = HttpMethod.Get,
            RequestUri = new Uri(jobId)
        })
        {
            request.Headers.Add("Ocp-Apim-Subscription-Key", apiKey);
            using (HttpResponseMessage response = await client.SendAsync(request))
            {
```

```
                if (!response.IsSuccessStatusCode)
                {
                    throw new HttpRequestException($"The remote service {jobId} responded with a {response.StatusCode} error code.");
                }
                string responseBody = await response.Content.ReadAsStringAsync();
                var responseObject = JObject.Parse(responseBody);
                return (responseObject.SelectToken("status").ToObject<string>(), responseObject);
            }
        }
    }
}


public static List<JToken> FindTokens(this JToken containerToken, string name)
{
    List<JToken> matches = new List<JToken>();
    FindTokens(containerToken, name, matches);
    return matches;
}

private static void FindTokens(JToken containerToken, string name, List<JToken> matches)
{
    if (containerToken.Type == JTokenType.Object)
    {
        foreach (JProperty child in containerToken.Children<JProperty>())
        {
            if (child.Name == name)
```

```
            {
                matches.Add(child.Value);
            }
            FindTokens(child.Value, name, matches);
        }
    }
    else if (containerToken.Type == JTokenType.Array)
    {
        foreach (JToken child in containerToken.Children())
        {
            FindTokens(child, name, matches);
        }
    }
  }
}
```